The Nature of Computation
Solution Manual

Cristopher Moore & Stephan Mertens

April 8, 2019

No problem is so big or so complicated that it can't be run away from!

Charles M. Schulz, *The Complete Peanuts*, 1959–1962

All references to equations, figures and sections refer to the book with the exception of references like (a), (b), ... that refer to equations within a solution.

This manual is work in progress. Please check www.nature-of-computation.org for a fresher version, indicated by the date on the title page.

Cristopher Moore and Stephan Mertens
Santa Fe and Magdeburg, 2014

Chapter 1

Prologue

1.1 Let $\delta(v)$ denote the degree of vertex v. Since any edge contributes to the degree count of two vertices, we have $\sum_v \delta(v) = 2|E|$ where $|E|$ is the number of edges. If V_e denotes the set of vertices with even degree and V_o denotes the set of vertices of odd degree we can write

$$2|E| = \sum_{v \in V_e} \delta(v) + \sum_{v \in V_e} \delta(v).$$

The first sum is even because all its summands are even. The second sum has to be even, too, but all its summands are odd. This implies that the number of summands must be even.

1.2 The pigeons are the vertices, and the holes are the degrees. The degree ranges from 0 (connected to nobody) to $n-1$ (connected to everyone else), and there are n vertices. It sounds like there are enough pigeons for all the holes—but if there is a vertex of degree zero then there can't be one of degree $n-1$, and vice versa. Thus there are only $n-1$ holes, ranging from 0 to $n-2$ or from 1 to $n-1$. Two pigeons must be in the same hole, so two vertices must have the same degree.

As an alternate proof, it suffices to prove the claim for connected graphs with at least two vertices. In such a graph, no vertex has degree 0, so there are again just $n-1$ possible degrees.

1.3 Let G be a connected graph with every vertex having even degree. Select any vertex and call it a. Select any edge incident to a and traverse it. Leave any subsequent vertex on an edge that has not been used and continue until forced to stop. The result is a walk W that begins in a. Whenever W passes through a vertex, it uses two edges: one for arrival, one for departure. So after traversal of W, each vertex still has an even number of unused edges (which may be zero). The only exception is vertex a, which we touched but didn't pass through. So vertex a is the only vertex that upon subsequent arrival may have no unused edge to leave it. Hence W must stop at a, i.e., W is closed. If W passes through all edges of G, we are done. If not, select the first vertex on W that has unused edges and call it b. Start the same procedure on b to get another closed path W' that is edge disjunctive with W. Then splice the two cycles at their crossing b as shown in Figure 1.8. The result is a new cycle that contains more edges than both its constituents. Repeat this procedure until the cycle contains all edges of G.

Now let G be a connected graph with exactly two odd vertices a and b. Form a new graph G' by adding a vertex z and edges (a,z) and (z,b). In G', every vertex has even degree. Hence G' has an Eulerian cycle.

Removing z and its two incident edges (a,z) and(z,b) from G' and that cycle, we get an Eulerian path in G from a to b.

1.5 Euler's argument, that every time we enter a vertex we have to leave it along a different edge, shows that every vertex must have an equal number of incoming and outgoing edges.

1.6 Only the octahedron is Eulerian, since the others have vertices of odd degree. All of them are Hamiltonian.

1.7 Color the vertices of the grid black and white like a checkerboard. The color alternates each time we take a step, so if there is a Hamiltonian cycle, the total number of vertices must be even. Thus at least one of m and n must be even. Conversely, if there are both even, there is a Hamiltonian cycle which weaves back and forth across the grid and then back along one edge.

1.8 The graph on the left is colored so that each edge connected a light vertex to a dark one. Formally, it is *bipartite*. Any Hamiltonian cycle would have to have an equal number of light and dark vertices, but there are an odd number of vertices, and one more dark one than light ones. More generally, no bipartite graph with an odd number of vertices can be Hamiltonian.

For the Petersen graph, we use the fact that it is highly symmetric. In particular, all the paths of four vertices are equivalent to each other. So, we can avoid most of the search tree by choosing one of these paths and trying to complete it. Let's take four vertices going clockwise around the outside of the graph. The fifth vertex in this cycle has to be on the Hamiltonian cycle, so it has to connect to the first or last vertex in the path of four. Then there is no way to visit all five of the inner vertices.

1.9 Let G be your graph. Construct a new graph G' by doubling each edge in G. All vertices in G' have even degree. According to Euler, G' has an Eulerian cycle which crosses every edge in G' exactly once. In G, this cycle crosses each edge exactly twice. Moreover, the shortest postman's tour can't repeat any edge three or more times, since we could form a loop from two of those repeats and remove it from the tour using moves like Figure 1.8.

Now consider the Chinese postman problem when G is a tree. Since the postman travels in a cycle, each vertex has at least one arrival and one departure edge. Let e be the edge by which we first leave v, and let e' be the edge by which we next arrive at v. If $e \neq e'$, then the part of the tour in between them forms a cycle in G, which a tree cannot have. Thus $e = e'$ and e is crossed twice, and the same argument holds for every edge in G.

In contrast, if G contains a cycle C we can make C part of the tour, build a postman tour for the rest of the graph (or the union of several such tours if G with C removed is not connected) and combine them with the move of Figure 1.8. Thus G has a tour where the edges on C are crossed only once.

Now consider a vertex v. Since the total number of steps in the tour entering and leaving it is even, the number of v's edges which are crossed twice is odd or even if v's degree is odd or even respectively. We can then take the set of all such edges and write it as a union of loops and paths. In the shortest possible tour, there are no loops, since we could traverse this loop once and remove it from the tour as we just discussed. Thus we are left with a set of paths whose endpoints have odd degree.

Finally, if we define a complete graph on the set of odd-degree vertices where the weight w_{ij} is the length of the shortest path from i to j, we can use Edmonds algorithm (see Note 5.6) to find the matching of minimum total weight. The length of the minimum postman tour is then the total weight of this matching plus the total number of edges.

1.10 There are many ways to solve this problem. Here's one. Suppose the original graph G has n vertices, and that the oracle says "yes" to the question of whether G is Hamiltonian. Go through the edges in any order. For each edge e, ask whether G would still be Hamiltonian if we removed e. If she says "yes," remove e from the graph. If she says "no", mark that edge as essential, and never ask about it again. Continue in this way until there are only n edges left—all of which are essential—which form a Hamiltonian cycle for G.

Since each edge is either removed are marked as essential when we ask about it, we ask about each edge at most once. Thus the total number of questions we ask (or drachmas we spend) is at most the number of edges, which in a simple graph is $O(n^2)$.

Chapter 2

The Basics

2.1 If she can upgrade once every two years, Brilliant Pebble should wait for the next upgrade if T is greater than four years, since this will let her graduate in $2 + T/2 < T$ years instead.

On the other hand, if we assume that processing speed is constantly improving and she can upgrade at any time, then waiting t years lets her graduate after a total time of $t + 2^{-t/2}T$. If $T > 2/\ln 2 \approx 2.89$, she can minimize the total time by starting her program at $t = 2(\log_2(T \ln 2) - 1)$.

2.2 If we extend Euclid's algorithm such that it computes the remainder and the quotient at each recursion, the sequence of equations reads

$$\begin{aligned} a &= q_0 b + r_0 \\ b &= q_1 r_0 + r_1 \\ &\;\vdots \\ r_{n-2} &= q_n r_{n-1} + 0\,, \end{aligned}$$

with $r_{n-1} = \gcd(a,b)$. This can be written as a product of 2×2 matrices:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} q_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b \\ r_0 \end{pmatrix} = \begin{pmatrix} q_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = \cdots = \prod_{i=0}^{n} \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{n-1} \\ 0 \end{pmatrix}$$

Let M denote the matrix

$$M = \prod_{i=0}^{n} \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}.$$

Note that $\det M = \pm 1$ because the determinant of each factor in the product is -1. Then

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \gcd(a,b) \\ 0 \end{pmatrix},$$

and a simple matrix inversion yields

$$\begin{pmatrix} \gcd(a,b) \\ 0 \end{pmatrix} = (\pm 1) \begin{pmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

The first row is (2.7).

For the second claim note that for $\gcd(a,b)=1$, the extended Euclidean algorithm yields x' and y' such that $ax+by=1$. Taking both sides of this equation modulo a yields $by = 1 \bmod a$. If $y \notin [0,a)$, we can add or subtract k multiples of a such that $y \mapsto y \pm ka \in [0,a)$.

2.3 This method of finding $a \bmod b$ will take a/b subtractions. If $a = 2^n$ and $b=2$, say, then $a/b = 2^{n-1} = \Omega(2^n)$.

2.4 The key thing to notice is that φ^ℓ grows exactly as fast as the Fibonacci equation asks: we have

$$\varphi^\ell = \varphi^{\ell-1} + \varphi^{\ell-2}$$

since dividing by both sides by $\varphi^{\ell-2}$ gives the quadratic equation $\varphi^2 = \varphi + 1$.

With that in mind, if we have $F_{\ell-1} \le B\varphi^{\ell-1}$ and $F_{\ell-2} \le B\varphi^{\ell-2}$, then

$$F_\ell = F_{\ell-1} + F_{\ell-2} \le B\varphi^{\ell-1} + B\varphi^{\ell-2} = B(\varphi^{\ell-1} + \varphi^{\ell-2}) = B\varphi^\ell\,.$$

Similarly, if $F_{\ell-1} \ge A\varphi^{\ell-1}$ and $F_{\ell-2} \ge A\varphi^{\ell-2}$, then $F_\ell \ge A\varphi^\ell$.

Therefore, if a bound of the form $A\varphi^\ell \le F_\ell \le B\varphi^\ell$ holds for $\ell-1$ and $\ell-2$, then it holds for ℓ, and by induction, all larger values of ℓ. All we need is a base case. Since $F_1 = F_2 = 1$, if we set $A = 1/\varphi^2$ and $B = 1/\varphi$ then we have $A\varphi^\ell \le F_\ell \le B\varphi^\ell$ for $\ell = 1$ and $\ell = 2$. Induction then shows that this holds for all $\ell \ge 1$.

Note that these bounds don't hold if we extend the Fibonacci sequence backwards!

2.5 Since $F_\ell = \Theta(\varphi^\ell)$, there are just $O(n)$ Fibonacci numbers between 1 and 10^n. So, we simply generate the Fibonacci numbers in order using (2.8) and compare each one to x. Each step is an n-digit addition, which we can do in poly(n) time. After $O(n)$ steps we either hit x and return "yes," or pass x and return "no."

2.7 One iteration of Euclid's algorithm maps the ratio b/a to $(a \bmod b)/b$. But $a = nb + (a \bmod b)$ for some $n \in \mathbb{N}$. Hence

$$\frac{a}{b} \bmod 1 = \left(n + \frac{a \bmod b}{b}\right) \bmod 1 = \frac{a \bmod b}{b}\,.$$

Therefore $x = b/a$ is mapped to $1/x \bmod 1 = g(x)$.

From $g(y) = 1/y - \lfloor 1/y \rfloor$ we get $g'(y) = -1/y^2$ for $y > 1$. The only solutions of $x = g(y)$ in the interval $(0,1)$ are $y = 1/(x+n)$ for $n = 1,2,3,\ldots$. Hence the stationary distribution $P(x)$ is the solution of

$$P(x) = \sum_{n=1}^{\infty} P\left(\frac{1}{x+n}\right) \frac{1}{(x+n)^2}\,.$$

This fixes $P(x)$ only up to a constant factor c. Plugging the ansatz $P(x) = c/(1+x)$ into the rhs gives

$$\begin{aligned} c\sum_{n=1}^{\infty}\left(\frac{1}{1+\frac{1}{x+n}}\right)\frac{1}{(x+n)^2} &= c\sum_{n=1}^{\infty}\left(\frac{1}{x+1+n}\right)\left(\frac{1}{x+n}\right) \\ &= c\sum_{n=1}^{\infty}\left(\frac{1}{x+n} - \frac{1}{x+1+n}\right) \\ &= \frac{c}{x+1} = P(x)\,. \end{aligned}$$

Normalization $\int_0^1 P(x)\,\mathrm{d}x = 1$ then gives $c = 1/\ln 2$.

2.12 Let's assume that $T(n) = Cn^\gamma$ for some constant C. Plugging this in to (2.10) gives

$$Cn^\gamma = aC(n/b)^\gamma$$

and dividing both sides by Cn^γ gives

$$b^\gamma = a$$

or $\gamma = \log_b a$. Plugging in $n = 1$ then gives $T(1) = C$.

2.18 If there are n vertices, the adjacency matrix takes n^2 bits. If there are m edges, each one takes $O(\log n)$ bits to describe its endpoints, for a total of $O(m \log n)$ bits. In a sparse graph where $m = O(n)$, the list of edges is $O(n \log n)$ bits long, much shorter than the adjacency matrix. In a dense graph with $m = \Theta(n^2)$, the list of edges is $\Theta(n^2 \log n)$ bits long, which is longer than the adjacency matrix by the factor $\log n$.

In a multigraph, the adjacency matrix can no longer consist just of 0s and 1s. Instead, A_{ij} should be the number of edges connecting i to j. If there are m edges total then A_{ij} is $O(\log m)$ bits long, so A's total length is $O(n^2 \log m)$. The list of edges is again $O(m \log n)$ if we naively list each copy of an edge separately.

2.19 If $f(n), g(n) = \text{poly}(n)$, then there are constants a, b such that $f(n) = O(n^a)$ and $g(n) = O(n^b)$. But then $f(g(n)) = O((n^b)^a) = O(n^{ab}) = \text{poly}(n)$ as well.

Now suppose we give A an input x of size n. Since A performs a polynomial number of steps, then the input y it sends to B can only be of polynomial size $n' = \text{poly}(n)$, and B returns its answer in time $\text{poly}(n') = \text{poly}(n)$.

On the other hand, if B's output is, say, twice the size of its input, then iterating B produces an output whose size grows exponentially with the number of iterations. For instance, consider this algorithm:

```
$A(x)$
begin
    for $t = 1$ to $\log_2 x$ do
        $x := B(x)$;
    end
    return $x$;
end
```

If B is the integer function $B(x) = x^2$ and x is n digits long, this squares x n times, giving the rather large function

$$A(x) = \underbrace{((x^2)^2)^{\cdots 2}}_{n \text{ times}} = x^{2^n}$$

which grows roughly like x^x, and the total running time of A is exponential.

2.20 If $f(n)$ is quasipolynomial with $k > 1$ and $g(n)$ is polynomial, there are many ways to show that $f(n) = \omega(g(n))$. One way is to consider the ratio of their logarithms:

$$\log \frac{f(n)}{g(n)} = \Theta(\log^k n) - O(\log n) .$$

Since this tends to ∞ as n does, $f(n)/g(n)$ tends to ∞ as well, and $f(n) = \omega(g(n))$.

Similarly, if $h(n) = 2^{n^c}$ for $c > 0$, then

$$\log \frac{f(n)}{h(n)} = \Theta(\log^k n) - \Theta(n^c)$$

tends to $-\infty$, so $f(n)/h(n)$ tends to 0 and $f(n) = o(h(n))$.

Finally, we show that the composition of two quasipolynomials is a quasipolynomial. If $f(n) = 2^{\Theta(\log^k n)}$ and $g(n) = 2^{\Theta(\log^\ell n)}$,

$$f(g(n)) = 2^{\Theta(\log^k g(n))} = 2^{\Theta(\log^{k\ell} n)} .$$

Chapter 3

Insights and Algorithms

3.2 Moving along an edge parallel to the ith axis corresponds to moving the ith smallest disk. The order in which we do this goes like this: $1,2,1,3,1,2,1,4,\ldots$. This gives a Hamiltonian path like the one shown in the figure. It's also like the order in which we flip bits in the Gray code (see Note 3.2). The 2^n vertices of the cube correspond to the 2^n states we go through, including the initial and final state, over the course doing the optimal solution.

3.9

$$\begin{aligned} An\ln n &= n + \frac{2}{n}\int_0^n Ax\ln x\,\mathrm{d}x \\ &= n + \frac{2A}{n}\left(\frac{n^2\ln n}{2} - \frac{n^2}{4}\right) \\ &= n + An\ln n - \frac{An}{2}\,, \end{aligned}$$

so $0 = n - An/2$ and $A = 2$.

3.11 Plugging the assumption $T(n) = A\,n\ln n$ into the equation and simplifying gives

$$A = \frac{1}{-\gamma\ln\gamma - (1-\gamma)\ln(1-\gamma)}\,.$$

The denominator $h(\gamma) = -\gamma\ln\gamma - (1-\gamma)\ln(1-\gamma)$ is often called the *entropy.* It represents the amount of information in a flip of a biased coin which comes up heads or tails with probability γ and $1-\gamma$ respectively. At $\gamma = 1/2$, we have $h(\gamma) = \ln 2$, corresponding to a single bit of information. Near $\gamma = 0$ or $\gamma = 1$, $h(\gamma)$ approaches 0, and A diverges.

3.12 If we can find the smallest divisor d of y, which is necessarily a prime, we can divide y by d and recurse until we have factored y completely. Since $d \geq 2$, this will take at most $\log_2 y = O(n)$ steps.

To find d, we note that $\gcd(x!, y) = 1$ if and only if $x < d$, and that $\gcd(x!, y) = \gcd(x! \bmod y, y)$. Since we can find the gcd of two n-digit numbers in polynomial time, we can tell whether $x < d$ with one call to MODULAR FACTORIAL.

Finally, if we can tell whether $x < d$ or not, we can locate d using binary search on the interval ranging from 2 to y. This takes $O(\log y) = O(n)$ steps. So, we can find d with $O(n)$ calls to MODULAR FACTORIAL, and factor y completely with a total of $O(n^2)$ calls.

3.13 We have $x! \bmod y = \Gamma(x-1) \bmod y$, so the question is whether we can compute $\Gamma(x) \bmod y$ in polynomial time for n-digit numbers x. Since $\Gamma(x) = (x-1)\Gamma(x-1)$, we can recursively change the least significant bit of x from 1 to 0. If $\Gamma(2x) = f(x)\Gamma(x)^2$ where $f(x) \bmod y$ is in P, then we can also recursively strip a 0 off the end of x's bit sequence, dividing x by 2. This reduces $\Gamma(x)$ to the base case $\Gamma(0)$ in $O(n)$ steps, each of which takes poly(n) time.

3.18 If we use naive recursion, then the number of function calls $T(\ell)$ obeys exactly the same equation as the Fibonacci numbers themselves, $T(\ell) = T(\ell-1) + T(\ell-2)$. Since the base case lets us return F_ℓ without recursion when $\ell = 1$ or $\ell = 2$, we also have $T(1) = T(2) = 1$. Thus $T(\ell) = F_\ell$ for all $n \ge 1$, and we showed in Problem 2.4 that F_ℓ grows exponentially, as $\Theta(\varphi^\ell)$ where φ is the golden ratio.

On the other hand, if we memorize each Fibonacci number the first time we calculate it, then in order to calculate F_ℓ we only need to calculate F_i once for each $i \le \ell$, giving a total of $\ell = O(\ell)$ function calls.

3.19 We prove (3.20) by induction on m. The base case $m = 1$ is the familiar recursion, $F_\ell = F_{\ell-1} + F_{\ell-2}$. Now suppose we have established (3.20) for $m-1$, i.e.,

$$F_\ell = F_m\, F_{\ell-m+1} + F_{m-1}\, F_{\ell-m}\,.$$

But $F_{\ell-m+1} = F_{\ell-m} + F_{\ell-m-1}$, so

$$\begin{aligned} F_\ell &= F_m(F_{\ell-m} + F_{\ell-m-1}) + F_{m-1}\, F_{\ell-m} \\ &= (F_m + F_{m-1})F_{\ell-m} + F_m\, F_{\ell-m-1} \\ &= F_{m+1}\, F_{\ell-m} + F_m\, F_{\ell-m-1}\,. \end{aligned}$$

So, by induction, we have (3.20) for all $m \ge 1$. Setting $m = \ell$ then gives (3.21).

Now suppose we wish to compute $F_\ell \bmod p$. According to (3.21), we can use a divide-and-conquer approach. Let $T(\ell)$ be the time its takes to compute F_ℓ. But if we use (3.21) directly, and ignore the ± 1s, we get

$$T(2\ell) = 2T(\ell)\,,$$

even if we ignore the time it takes to multiply and add the terms in (3.21). The solution to this is $T(\ell) = \Theta(\ell)$. But we want a running time which is polynomial in $n = \log_2 \ell$, not in ℓ.

The problem is that this direct recursion recomputes some Fibonacci numbers many times. A better approach is to compute F_ℓ and $F_{\ell+1}$ simultaneously. Using (3.21) we can reduce the problem of computing the pair $(F_{2\ell}, F_{2\ell+1})$ to that of computing $(F_\ell, F_{\ell+1})$ (using the fact that $F_{\ell-1} = F_{\ell+1} - F_\ell$), along with some addition and multiplication of n-bit integers. If the time it takes to do this for an n-bit ℓ is $f(n)$, then since we can multiply two n-bit integers in $O(n^2)$ time (or even less), we get the recurrence

$$f(n) = f(n-1) + O(n^2)\,.$$

to which the solution is $T(n) = O(n^3)$.

3.20 If there are n words $w_1, \ldots, w_n$, define a graph with $n+1$ vertices $v_0, v_1, \ldots, v_n$ where v_i corresponds to placing a line break after w_i. For each pair i, j with $i < j$, there is an edge from v_i to v_j with weight equal to the cost $c(i+1, j)$ of putting the ith through the jth words on a single line. Then the optimal choice of line breaks corresponds to the shortest path from v_0 to v_n.

3.22 Let's first find the length of the longest increasing subsequence, or l.i.s. for short, as opposed to finding the subsequence itself. We use dynamic programming. Let $f(i, t)$ be the length of the l.i.s. of $s_i, s_{i+1}, \ldots, s_n$ such that its first element is greater than t. Let's assume that the s_i are nonnegative, so that the l.i.s. of the entire list has length $f(1, 0)$.

Now since s_i is either in the l.i.s. of $s_i, \ldots, s_n$ or not, and since it only counts towards $f(i, t)$ if $s_i > t$, we have

$$f(i,t) = \begin{cases} \max\Big(1 + f(i, s_i),\, f(i+1, t)\Big) & \text{if } s_i > t \\ f(i+1, t) & \text{if } s_i \le t \end{cases}$$

If we calculate $f(1, 0)$ recursively, we will ask about at most n^2 different values of $f(i, t)$, since $1 \le i \le n$ and t will always be one of the s_i. So while this recursive function would take exponential time if we recalculate $f(i, t)$ every time, if we memorize the values of $f(i, t)$ we have already calculated it will take $O(n^2)$ time.

We can modify the function f so that it returns the l.i.s. instead of its length. Just replace $1 + f(i, s_i)$ with $s_i \cdot f(i, s_i)$ where $\cdot$ denotes concatenation, and if a and b are strings let max(a, b) return the longer one.

3.23 Suppose that $i < j$. If $s_i < s_j$, then $a_i < a_j$ since we can add s_j to any increasing subsequence ending with s_i. Similarly, if $s_i > s_j$ then $b_i < b_j$. Therefore, we cannot have both $a_i = a_j$ and $b_i = b_j$, and all n of the pairs (a_i, b_i) for $1 \le i \le n$ must be distinct. However, if the longest increasing or decreasing subsequence is of length k, then there are only k^2 possible pairs (a_i, b_i). Thus we have n pigeons trying to roost in k^2 holes, and $n \le k^2$. Inverting this gives $k \ge \lceil \sqrt{n} \rceil$.

3.25 As in Problem 3.22, it's handy to first think about calculating the weight of the maximum independent set, and then modify the algorithm so that it finds the set itself. Choose one vertex r, call it the root, and orient the tree so that each vertex v has a subtree T_v consisting of it and the vertices below it. Let $f(v)$ be the weight of the maximum independent set of T_v; then our goal to calculate $f(r)$.

It will help to define another function: let $f_0(v)$ be the maximum independent set in T_v *that doesn't include* v. Then we can write both functions recursively in terms of each other. Since the maximum independent set of T_v either includes or excludes v, and if it includes v it must exclude all of v's children, we have

$$f(v) = \max\Big(w(v) + \sum_u f_0(u),\, \sum_u f(u)\Big)$$

where the sum ranges over all the children u of v. Similarly, we have

$$f_0(v) = \sum_u f(u).$$

Each of these functions only needs to be calculated once for each vertex v, so if we remember the values $f(v)$, $f_0(v)$ we have already calculated we can obtain $f(r)$ in polynomial time.

Then, if we want to function to return the maximum independent set rather than its weight, we can replace $w(v)+\sum_u f_0(u)$ with $v \cup \bigcup_u f_0(u)$ and so on.

3.26 Let's define some notation. Let T denote the entire tree, let T_i be the subtree rooted at the ith child of the root, and let T_{ij} be the subtree rooted at the jth child of the ith child (who is one of the root's grandchildren). Let $M(T)$ be the weight of the maximum-weight matching of T.

Let w_i be the weight of the edge from the root to the ith child. If $M(T)$ includes this edge, then it also includes $M(T_k)$ for all the other children $k \neq i$, and $M(T_{ij})$ for all j. Alternately, $M(T)$ might not include any of the root's edges, in which case it includes $M(T_i)$ for all children i. Thus

$$M(T) = \max\left(\max_i \left(w_i + \sum_{k \neq i} M(T_k) + \sum_j M(T_{ij}) \right), \sum_i M(T_i) \right).$$

The base case, of course, is $M(T) = 0$ if T is rooted at a leaf with no children. If we want the matching instead of its weight, we return instead whichever matching maximizes this expression.

There are n vertices in the tree. Thus we just need to compute M on n different subtrees, and this algorithm runs in poly(n) time.

3.27 To reduce to MAX-WEIGHT INDEPENDENT SET, define a graph with n vertices, each of which represents an interval, and draw an edge between any two intervals that overlap. Then a set of disjoint intervals is an independent set. Finally, give the vertex corresponding to each interval $[x_i, y_i]$ a weight equal to its price v_i.

To show that a greedy algorithm fails, suppose we have the three intervals $[0, 0.49]$, $[0.48, 0.52]$, and $[0.51, 1]$ with prices of \$51, \$100, and \$51 respectively. Selling $[0.48, 0.52]$ looks good, since it has both the highest price and the smallest length. But it would be better to sell the other two intervals instead, for a total of \$102 instead of \$100.

For a dynamic programming algorithm, sort the intervals $[x_i, y_i]$ from left to right, in increasing order of x_i. Define $f(i)$ as the largest total price of any independent subset of the intervals $[x_i, y_i], \ldots, [x_n, y_n]$. Either we should sell $[x_i, y_i]$, or we shouldn't. If we do, we gain v_i, but we must forgo all the intervals overlapping with it. Let $s(i)$ denote the index of the first interval which doesn't overlap with $[x_i, y_i]$, i.e., the smallest $j > i$ such that $x_j > y_i$. (We define $s(i) = n+1$ if there is no such j, and set $f(n+1) = 0$.) Then

$$f(i) = \max\big(v_i + f(s(i)), f(i+1)\big).$$

There are just n subproblems, namely $f(i)$ for each i between 1 and n, and the answer to the entire problem is given by $f(1)$.

3.28 Since a triangle of side ℓ consists of three "subtriangles" of side $\ell/2$, and since these share three vertices, the number of vertices $n(\ell)$ obeys the equation

$$n(\ell) = 3n(\ell/2) - 3$$

with the base case $n(2) = 3$. When ℓ is large the additive term -3 is negligible, and the asymptotic behavior is $n = \Theta(\ell^{\log_2 3})$ as in Problem 2.12.

To solve MAX-WEIGHT INDEPENDENT SET, we again use dynamic programming. Just as choosing whether or not to include the root of a tree in an independent set breaks the tree into independent subtrees, choosing whether or not to include each of these three shared vertices breaks the triangle into three independent subtriangles. For each subtriangle, we need to solve up to 8 problems, depending on whether its corners (which it shares with other subtriangles) are included or not. The total number of different subproblems $T(\ell)$ in a triangle of side ℓ is thus at most 8 plus the total number of subproblems for its three subtriangles, or

$$T(\ell) \leq 3T(\ell/2)+8\,.$$

Since this again gives $T(\ell)=\Theta(\ell^{\log_2 3})=\Theta(n)$, the total number of subproblems is linear in the number of vertices. Finally, since each problem can be solved by solving a constant number of subproblems, dynamic programming works in polynomial time.

3.29 We use dynamic programming again. We start with the top row, choosing which vertices to include in the independent set, or which edges that touch the top row are included in the matching. By removing the neighbors of these vertices, or the endpoints of these edges in the next row, we get a problem with one fewer row, where some vertices in the top boundary are forbidden. There are $\sqrt{n}\,2^{\sqrt{n}}=2^{O(\sqrt{n})}$ such subproblems.

Alternately, by drawing horizontal and vertical lines through the center of a square, and choosing which of the vertices on these lines will be included in the independent set, or which edges across these lines will be included in the matching, we can break the square into four independent "subsquares," each of which has side $\sqrt{n}/2$. These in turn can be broken into subsquares of side $\sqrt{n}/4$, and so on.

The total number of subsquares of all sizes is easily shown to be linear in n. However, some of the vertices on the perimeter of the subsquare are forbidden. Since there are $O(\sqrt{n})$ vertices on the perimeter, this again gives $2^{O(\sqrt{n})}$ different subproblems that have to be solved.

On a rectangle, we can achieve a running time of $2^{O(\text{height})}$ or $2^{O(\text{width})}$, whichever is smaller. On a cube we need to set the vertices on a square face, or on a square cut; this gives $2^{O(n^{2/3})}$, and more generally $2^{O(n^{1-1/d})}$ for a d-dimensional hypercube which is $n^{1/d}$ on a side.

The lesson here is that the running time of dynamic programming grows exponentially as a function of the size of the boundary between the subproblems. For a line, a tree, or a Sierpiński triangle, this boundary has constant size, but on a two-dimensional lattice it grows like the diameter $\sqrt{n}$.

3.30 We use dynamic programming. Recursively, we have $A \rightsquigarrow w$ either if there are nonempty strings u,v and variables B,C such that $w=uv$, $B \rightsquigarrow u$, $C \rightsquigarrow v$, and $(A \to BC) \in R$—or, in the base case, if w is a single symbol and $(A \to w) \in R$.

If we apply this recursive strategy, every subproblem we encounter will ask whether $A \rightsquigarrow u$ for some substring u of v. If $|w|=n$, there are $O(n^2)$ such substrings, so the number of different subproblems is $O(n^2|V|)$. Since V is given as part of the input, this is polynomial in the total size of the input.

3.31 At any point in the course of a depth-first exhaustive search, we have already visited some set of vertices S, not including our current position v. Our path so far can be completed to a Hamiltonian path if and only if there is a Hamiltonian path, starting at v, through the subgraph consisting of the other vertices $T=V-S$. Since there are 2^n possible subsets T and n possible vertices v, there are at most $2^n n$ subproblems of this form. Using dynamic programming—which, by the way, requires an exponential amount of memory in this case—we only need to solve each of these subproblems once.

To be precise, let $\texttt{path}(T,v)$ denote the statement that we can visit all of T starting at v. To solve HAMILTONIAN PATH, we want to compute $\texttt{path}(V,v)$, with v ranging over all the vertices. We then compute $\texttt{path}(T,v)$ recursively. The base case is

$$\texttt{path}(\emptyset,v)=\texttt{true},$$

and the recursive step is

$$\texttt{path}(T,v)=\bigvee_{w\in T:(v,w)\in E}\texttt{path}(T-\{v\},w). \tag{a}$$

If we have already computed $\texttt{path}(T-\{v\},w)$ for all w, computing $\texttt{path}(T,v)$ just requires a **for** loop over w, which takes $O(n)$ time. Thus the total running time is $O(n)$ times the number of subproblems, which is $2^n\,\mathrm{poly}(n)$. But this implies that we store and reuse the values of $\texttt{path}(T,v)$ for every subset $T\subseteq V$ and every v that we calculate along the way, which requires an exponential amount of memory.

Note that if we actually calculate (instead of look up) each term the right hand side of (a), the recursion for the running time is essentially $T(n)=n\,T(n-1)$, which brings us back to $T=O(n!)$.

3.33 There are lots of ways to prove this. One is to note that there are two ways to get from 1 to 2 in ℓ steps. You can get from 1 to 2 in $\ell-1$ steps, and then stay at 2. Or, you can get from 1 to 1 in $\ell-1$ steps, and then move to 2. In other words,

$$A^{\ell}_{1,2}=A^{\ell-1}_{1,2}+A^{\ell-1}_{1,1}.$$

On the other hand, the only way to get from 1 to 1 in $\ell-1$ steps is to get from 1 to 2 in $\ell-2$ steps, and then move to 1. So

$$A^{\ell-1}_{1,1}=A^{\ell-2}_{1,2},$$

and

$$A^{\ell}_{1,2}=A^{\ell-1}_{1,2}+A^{\ell-2}_{1,2}.$$

We can say all this at once, by multiplying the top row of $A^{\ell-1}$ by A:

$$\left(A^{\ell}_{1,1},A^{\ell}_{1,2}\right)=\left(A^{\ell-1}_{1,1},A^{\ell-1}_{1,2}\right)\begin{pmatrix}0&1\\1&1\end{pmatrix}=\left(A^{\ell-1}_{1,2},A^{\ell-1}_{1,1}+A^{\ell-1}_{1,2}\right).$$

This $A^{\ell}_{1,2}$ follows the Fibonacci recurrence. To show that it equals F_ℓ, we just need to check the base case, that it is 1 when $\ell=1$ or 2.

When we diagonalize A, we find that it has two eigenvalues, $\varphi=(1+\sqrt{5})/2$ and $-1/\varphi=(1-\sqrt{5})/2$, with eigenvectors $v_1=(1,\varphi)$ and $v_2=(-\varphi,1)$. We can write

$$A^{\ell}_{1,2}=\left(1,0\right)A^{\ell}\begin{pmatrix}0\\1\end{pmatrix}. \tag{a}$$

But we can write $(1,0)$ and $(0,1)$ as linear combinations of v_1 and v_2,

$$(1,0)=\frac{v_1-\varphi v_2}{\varphi^2+1}\quad\text{and}\quad(0,1)=\frac{\varphi v_1+v_2}{\varphi^2+1}.$$

Plugging this into (a) gives $A^{\ell}_{1,2} = a\varphi^{\ell} + b(-1/\varphi)^{\ell}$ where

$$a = \frac{\varphi}{\varphi^2+1} \quad \text{and} \quad b = -\frac{\varphi}{\varphi^2+1}\,.$$

We can also find a and b directly by solving two linear equations,

$$F_0 = 0 = a + b$$
$$F_1 = 1 = a\varphi - b/\varphi\,.$$

3.34 We will prove the following loop invariant by induction: after running the outer loop with a given value of k, B_{ij} is the length of the shortest path from i to j which only passes through vertices in the set $\{1,\ldots,k\}$.

The base case, as usual, is trivial: initially $k = 0$, and $B_{ij} = W_{ij}$ is the length of the shortest path from i to j which passes through no other vertices, namely 0 if $i = j$, the weight W_{ij} of the edge from i to j if there is one, and $+\infty$ otherwise.

For the induction step, suppose the loop invariant holds for $k-1$. The shortest path from i to j that passes through vertices in $\{1,\ldots,k\}$ either passes through k or it doesn't. If it doesn't, then B_{ij} keeps the value it had for $k-1$. If it does, then the paths from i to k and from k to j only pass through vertices in the set $\{1,\ldots,k-1\}$, so B_{ij} becomes $B_{ik} + B_{kj}$.

Finally, if $k = n$ then B_{ij} is the length of the shortest path from i to j which passes through any set of vertices, and we're done.

3.35 The bottleneck $b(i,j)$ is the maximum, over all k, of the bottleneck of a path that goes from i to j through k. This is the minimum of $b(i,k)$ and $b(k,j)$, so our "squaring" operation looks like

$$B_{ij} \mapsto \max_k \min(B_{ik}, B_{kj})\,.$$

This gives a one-line change to the pseudocode from Section 3.4.3,

$$B_{ij}(m) := \max\bigl(B_{ij}(m), \min(B_{ik}(m-1), B_{kj}(m-1))\bigr)\,.$$

3.36 Let L denote the list of all vertices that require exactly one more incident edge upon completion of the tree. Initially, L contains all the leafs of the tree, i.e., all vertices that are not listed in P_n. The following procedure then reconstructs the tree. Iterate for $j = 1,\ldots,n-2$:

1. Connect p_j to the smallest value in L.
2. Remove the smalles value from L.
3. If $p_j \neq p_{j+k-1}$ for all $t = 1,\ldots n-2-j$, i.e., if p_j has now become a vertex with exactly one edge missing, add p_j to L.

Note that through this iteration every vertex has appeared exactly once in L, but we removed only $n-2$ entries. Hence the final L contains exactly two vertices longing for an edge. Connecting these two vertices completes the tree.

3.38 Suppose that T is the minimum spanning tree, and that it includes e. I claim that there is some edge e' in C, which is not included in T, such that the unique path in T from one of the endpoints of e' to the other goes through e. Otherwise, it would be possible to get from one endpoint of e to the other without using e, and T would contain a cycle. If we add e' and remove e, we again have a spanning tree; but since e' is lighter than e, this new spanning tree is lighter than T, which is a contradiction.

So, our algorithm starts with the entire graph. We look for cycles, for instance by removing a candidate edge and asking whether we can still get from one of its endpoints to the other. We then find the heaviest edge on that cycle, remove it, and iterate until there are no cycles left. This is less efficient than Prim's algorithm, but it still runs in polynomial time.

3.42 Associate each edge (i,j) with a vector v_{ij} such that $v_i = +1$, $v_j = -1$, and all other components are zero. The set of linear dependencies between these vectors, i.e., the linear combinations that sum to zero, is generated by cycles, so a set of edges is linearly independent if and only if it is a forest.

The subspace V is spanned by vectors of the form v_{ij} for all pairs of vertices i and j, and we can obtain v_{ij} by summing the edges along a path from i to j (with each edge given a weight of $+1$ or -1). A spanning tree spans V since there is such a path between any pair of vertices.

3.44 First consider the case where d stays the same. When we increase the flow as much as possible along δ, we use the edge $e \in \delta$ with the smallest $c_f(e)$ completely, removing it from G_f. This will decreases $|F|$ by at least one, unless e is a forward edge and the reverse edge $\overline{e}$ is now in F. We will show this is impossible.

Let $d(i,j)$ denote the length of the shortest directed path from i to j in G_f. For any $e = (u,v) \in F$, we have $d(s,v) = d(s,u)+1$, and moreover $d = d(s,t) = d(s,u)+d(v,t)+1$. In order for $\overline{e} = (v,u)$ to be added to F, these distances have to change so that $d(s,u) = d(s,v)+1$. However, for any w the distances $d(s,w)$ and $d(w,t)$ monotonically increase throughout the algorithm, since adding reverse edges can never decrease them. Therefore, if at some later stage we have $\overline{e} \in F$, then both $d(s,u)$ and d must have increased by at least 2.

Thus each step either decreases $|F|$ or increases d. Since $|F|$ is at most $|E|$ and d is at most $|V|$, the total number of improvements we can make until there are no paths left from s to t, at which point the flow is maximal, is $|E||V|$.

3.46 If we add an edge e to the current spanning tree T, we get a graph $T \cup \{e\}$ with a single cycle. Let e' be the heaviest edge in this cycle. If $w_{e'} > w_e$, then removing e' from $T \cup \{e\}$ gives a new tree T' which is lighter than T. Continue until, for every edge $e \notin T$, e is one of the heaviest edges in the cycle in $T \cup \{e\}$. At that point T is one of the lightest possible trees.

Alternately, pick an edge e. Removing it divides T into two parts. Find the lightest edge e' that joins them, and add e' to T. Repeat until every edge in T is one of the lightest edges connecting the two parts of the tree that would result from removing it.

3.47 One direction is easy. If there is a subset S on the left which is connected to fewer than $|S|$ vertices on the right, then by the pigeonhole principle there is no way to find partners for all the elements of S. Therefore if there is a perfect matching, every subset on the left is connected to a subset on the right which is at least at large.

In the other direction, suppose there is no perfect matching. Then according to the reduction of Figure 3.23, the MAX FLOW from s to t is less than n. By duality, the MIN CUT is also less than n, so there is some cut consisting of $c < n$ edges which separates s from t.

We claim that these edges might as well be among those leading out of s or into t in Figure 3.23, instead of in the middle layer of edges from the original bipartite graph. To see this, suppose the cut includes an edge (i, j) where i is one of the n vertices on the left and j is one of the n vertices on the right. This edge only blocks one path from s to t: namely, the path $s \to i \to j \to t$. If we replace (i, j) with (s, i) or (j, t), we still block this path, and perhaps others as well.

So, there is a cut consisting of x edges leading from s to some set of vertices on the left, and y edges leading from some set of vertices on the right to t, where $x + y < n$. Call these sets X and Y respectively, where $|X| = x$ and $|Y| = y$.

Now, for these edges to block all paths from s to t, there must be no edges from $\overline{X}$ to $\overline{Y}$. Thus if $S = \overline{X}$, the set T of vertices on the right that S is connected to is a subset of Y. But since $y < n - x$, we have $|T| \le |Y| = y < n - x = |S|$. So, if there is no perfect matching, there is a subset on the left connected to a smaller subset on the right.

3.48 Reduce the problem to MAX FLOW as follows: give each edge a capacity of 1, and ask whether there is a flow from s to t with value k.

3.50 We add two additional vertices, v_+ and v_-, and ask for the minimum cut that separates them. We think of grouping each s_i with v_+ or v_- depending on whether $s_i = +1$ or -1. Then we put edges connecting v_+ and v_- with each s_i, with weights corresponding to s_i's energy if it disagrees with its external field h_i. Finally, we put edges between s_i and s_j, with weights corresponding to the energy of these sites if they disagree with each other.

Let's define a baseline energy, which is as low as possible:

$$E_0 = -\sum_{ij} J_{ij} - \sum_i |h_i| .$$

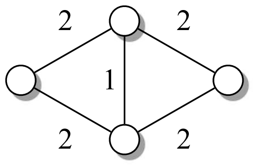
This is the energy the spin glass would have if every vertex was aligned with its external field, and if they were all aligned with each other. (In general, this is impossible, but clearly the energy is always at least E_0.)

Flipping s_i so that it disagrees with its external field increases E by $2|h_i|$. Similarly, flipping s_i or s_j so that they disagree with each other increases E by $2J_{ij}$. So, we define the weights of our MIN CUT problem as follows:

$$\begin{aligned} w(v_+, s_i) &= \begin{cases} 2h_i & \text{if } h_i > 0 \\ 0 & \text{if } h_i \le 0 \end{cases} \\ w(v_-, s_i) &= \begin{cases} 2|h_i| & \text{if } h_i < 0 \\ 0 & \text{if } h_i \ge 0 \end{cases} \\ w(s_i, s_j) &= 2J_{ij} . \end{aligned}$$

In the ferromagnetic case, where $J_{ij} \ge 0$, these weights are all nonnegative. Finally, if w is the weight of the minimum cut, the minimum energy is $E = E_0 + w$.

3.51 The greedy strategy fails. Consider the following graph:



The minimal 2-connected subgraph consists of the four outer edges, and doesn't include the lighter inner edge at all. Therefore, any greedy algorithm that starts by adding the lightest edge fails.

More generally, it seems unlikely that this problem is in P, because we can reduce HAMILTONIAN PATH to it. Simply consider the case where every edge in G has weight 1. A 2-connected subgraph of total weight n must be a Hamiltonian cycle. So, G is Hamiltonian if and only if its minimal 2-connected subgraph has weight n. This shows that

$$\text{HAMILTONIAN PATH} \leq \text{MINIMUM 2-CONNECTED SPANNING SUBGRAPH}\,.$$

So, this problem is at least as hard as HAMILTONIAN PATH, and isn't in P unless HAMILTONIAN PATH is.

Chapter 4

Needles in a Haystack: the Class NP

4.1 Given a formula $\phi(x_1,\ldots,x_n)$, setting $x_1 = \texttt{true}$ removes the clauses in which x_1 appears positively, and shortens those in which it appears negatively. Let $\phi[x_1 = \texttt{true}]$ denote the resulting formula on $x_2,\ldots,x_n$. Define $\phi[x_1 = \texttt{false}]$ similarly. We start by asking the oracle whether ϕ is satisfiable. If she says yes, we ask her about $\phi[x = \texttt{true}]$ and $\phi[x = \texttt{false}]$. At least one of these will be satisfiable, so with at most two queries we learn a value of x_1 for which a satisfiable assignment exists. If the resulting formula is ϕ', we then ask about $\phi'[x_2 = \texttt{true}]$ and $\phi'[x_2 = \texttt{false}]$, and so on. After at most $2n$ queries, we have found a satisfying assignment.

4.2 If we have a partial k-coloring c of G, such that $c(v) \in \{1,\ldots,k\}$ for some vertices v and $c(v)$ is undefined for others, our goal is to define a new G' that is k-colorable if and only if c can be completed to a proper k-coloring. Then we can add one vertex at a time to c, by trying its k possible colors.

Here's one way to do this. Add k new vertices, labeled $1,\ldots,k$, forming a complete graph K. Then construct G' from $G \cup K$ as follows. For each vertex v which is colored in c, merge v with vertex $c(v)$ in K. (Alternately, draw an edge between v and the $k-1$ vertices in K other than $c(v)$.) This forces all the vertices in G with a given $c(v)$ to have the same color (since they are now the same vertex), while forcing those with different $c(v)$ to have different colors (since there is an edge between them in K). If G' is k-colorable, then G has a k-coloring which agrees with c.

Another approach is to start with the graph G, choose two vertices which are not adjacent, and try to merge them together. If G is still k-colorable after we do this, it means there is a coloring where these two vertices are the same color. If not, these vertices have to be different colors (perhaps because of decisions we made earlier) so we should un-merge them. Keep merging pairs of vertices until only k vertices are left. Then the vertices of G which became the ith of these k vertices can be colored i.

4.3 If the graph contains triangular faces only, the coloring process is trivial. Start with any face; coloring its three vertices requires all three colors, and the colors of the rest of the vertices are uniquely determined by the coloring of these initial three vertices. We don't even need to solve PLANAR GRAPH 3-COLORING in order to 3-color such a graph or to see that this is impossible. The problem then remains to triangulate a given planar graph in a way that preserves its 3-colorability.

So, let's assume that our graph has at least one face with $p > 3$ vertices. Select any pair of non-adjacent vertices u and v of this face and connect them by an edge, cutting the face into two faces, each of them with fewer vertices. Feed this new graph to PLANAR GRAPH 3-COLORING. If the new graph is still 3-colorable, we leave the edge in place. Otherwise, we know that u and v must have the same color in any proper 3-coloring. In this case we can safely merge u and v. Doing so divides this face into smaller ones (where a face with 2 vertices is just an edge) and doesn't increase the number of vertices in any other face.

We iterate this procedure until we get a graph G' that is completely triangulated. The coloring of G' can easily be transformed into a coloring of the original graph, provided we have maintained a list of which vertices have been merged during the process.

For 4-coloring, this does not work. First of all, the "power" to solve PLANAR GRAPH 4-COLORING is not helpful because we already know that the answer is always "yes." So, we can triangulate the entire graph without losing 4-colorability; but the oracle's repeated "yes" answers don't help us find a coloring of the triangulated graph. There are in fact polynomial-time algorithms for finding a proper 4-coloring of a planar graph, but they are far more complicated, and are closely related to the proof of the Four Color Theorem.

4.6 Using Problem 4.5 we assume that only three countries meet at each corner of the map. In that case, the graph of borders on the map (not to be confused with the dual graph) is 3-regular. Suppose we have a 3-edge-coloring of this graph, so that each border between two colors is given one of three colors. We claim that this tells us how to 4-color the countries of the map so that no two countries have the same color if they share a border.

The idea is to use the edge coloring to tell us how to change color when we cross a border. Call the 4 country colors 1, 2, 3, and 4, and call the 3 border colors A, B, and C. Start by choosing one country and coloring it 1. Then each time you cross a border, switch country colors according to the border color as follows:

$$\begin{array}{ll} A & 1 \longleftrightarrow 2, 3 \longleftrightarrow 4 \\ B & 1 \longleftrightarrow 3, 2 \longleftrightarrow 4 \\ C & 1 \longleftrightarrow 4, 2 \longleftrightarrow 3 \end{array}$$

It's easy to check that these color changes are consistent, in the sense that if we start in one country and go around a vertex where three borders meet, we get back to the same country color we had before. The only danger is if the graph of borders has a bridge, but this would correspond to the outermost country bordering itself. Therefore, if 3-regular bridgeless graphs are 3-edge-colorable, maps are 4-colorable, and Tait's conjecture is an alternate statement of the Four Color Theorem.

4.9 Suppose there is a 2-SAT formula ϕ that simulates $(x \vee y \vee z)$ in this way. Our discussion in Section 4.2.2 shows that there is a satisfying assignment where a given set of variables takes a given set of truth values if and only if there are no paths in the directed graph $G(\phi)$ which cause a contradiction. Now, since ϕ has a satisfying assignment where $x = y = \texttt{false}$, there is no path in $G(\phi)$ from $\overline{x}$ to y or from $\overline{y}$ to x. Similarly, since ϕ has satisfying assignments where $x = z = \texttt{false}$ or $y = z = \texttt{false}$, there are no paths from $\overline{x}$ to z, from $\overline{z}$ to x, from $\overline{z}$ to y, or from $\overline{y}$ to z. But that means that there is also a satisfying assignment where $x = y = z = \texttt{false}$, so ϕ fails to simulate this 3-SAT clause.

However, the lack of such a gadget does not mean that 3-SAT is outside P. In principle there could be a reduction from 3-SAT to 2-SAT, and therefore a polynomial-time algorithm for 3-SAT, that uses some global information about the 3-SAT formula, as opposed to locally replacing each clause with a gadget.

4.10 DNF-SAT is in P because, in order to satisfy the formula, we just have to satisfy a single clause. A DNF clause is satisfiable as long as it doesn't contain a contradictory pair of literals, and we can scan the formula in $O(n)$ time to tell if it contains a satisfiable clause.

We can change a CNF formula to a DNF formula using the distributive law. For instance,

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) = (x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \wedge y_2),$$

just as $(a+b)(c+d) = ac + ad + bc + bd$. However, given a CNF formula with m clauses and k variables per clause, this gives a DNF formula with k^m clauses, each with m variables. For instance, the example formula would become a DNF formula with 2^n clauses, where each one consists of the AND of n variables, one from each of the n original clauses:

$$\bigvee_{\{z_i\}} \left(\bigwedge_{i=1}^{n} z_i \right)$$

where the $\bigvee$ is over all sets of variables z_i such that $z_i = x_i$ or y_i for each i.

Therefore, converting CNF formulas to DNF ones can increase their size exponentially, and this conversion does not qualify as a polynomial-time reduction. We cannot say that

$$\text{CNF-SAT} \le \text{DNF-SAT},$$

so there is no contradiction if DNF-SAT is in P but CNF-SAT is not.

4.11 If there is a perfect matching between clauses and a subset of the variables—that is, if we can match each clause with one of its variables, so that no variable is matched with more than one clause—then we can satisfy each clause by setting its partner variable. Hall's Theorem states that such a matching exists if and only if every subset of s clauses contains at least s variables. Suppose this is not the case: that some subset of s clauses contains only $t < s$ variables. Then since each clause has k variables, by the pigeonhole principle at least one of these variables must have degree $k+1$. Conversely, if no variable has degree greater than k then such a matching exists and the formula is satisfiable. Moreover, we can find it in polynomial time by reducing to MAX FLOW.

4.12 This is a classic example of the pigeonhole principle. There are $2^{10} = 1024$ different subsets, but their totals all lie in the range 0 to 1000. Since $1024 > 1001$, it is impossible for each of the subsets to have a different total. Therefore, there must be two sets A and B with the same total. Removing $A \cap B$ from both of them makes them disjoint, while preserving the fact that they have the same total.

4.13 Suppose we have an instance of SUBSET SUM with total weight $W = \sum_i x_i$ and target weight t. We will transform this to an instance of INTEGER PARTITIONING by adding an additional weight y so that we can put half of $W + y$ on each side if and only if there is a subset A with total weight t. There are three cases. If $t = W/2$, we're already done. If $t < W/2$, we set $y = W - 2t$: then $t + y = (W + y)/2$, so one side consists of $A \cup \{y\}$ and the other of $\overline{A}$. If $t > W/2$, we set $y = 2t - W$: then $t = (W + y)/2$, so one side consists of A and the other of $\overline{A} \cup \{y\}$.

4.14 Let A be the subset, if there is one, which sums to t. Let a_j be the largest element of the superincreasing sequence such that $a_j \le t$. None of the numbers a_i with $i > j$ can be in A since they are all larger than t. Similarly, all the numbers a_i with $i < j$ sum up to some value less than a_j, so they alone can never sum up to t. Hence a_j must be an element of A. Replace t by $t - a_j$, and remove a_j from the sequence. Iterate until either $t = 0$, in which case we have found a solution, or until $a_j > t$ for all j, in which case no solution exists. If $S = \{1, 2, 4, 8, \ldots\}$, this algorithm produces the subset corresponding to the binary expansion of t.

4.15 Divide S into S_1 and S_2 where $|S_1| = |S_2| = \ell/2$. Each one has $2^{\ell/2}$ subsets. Computing all their totals and sorting them in increasing order takes $2^{\ell/2}\,\mathrm{poly}(\ell)$ time. Then go through the list of totals of subsets of S_1. For each total t_1, check to see if S_2 has a subset with total $t - t_1$.

4.16 We will show that the size of the smallest vertex cover equals the size of the maximum matching. This reduces these problems to MAX BIPARTITE MATCHING, and shows that they are in P.

First, let S be a vertex cover, and let M be a matching. At least one endpoint of each edge in M must be included in S, so $|S| \geq |M|$. Thus the minimum vertex cover is at least as large as the maximum matching, or $\min |S| \geq \max |M|$.

However, showing that $\min |S| = \max |M|$ takes more work. There are several ways to do this, but we will use the reduction from MAX BIPARTITE MATCHING to MAX FLOW and the duality between MAX FLOW and MIN CUT. Add vertices s and t to the left and right sides of the graph as in Figure 3.23 on page 75. As in the proof of Hall's theorem in Problem 3.47, there is a minimum cut consisting of edges connecting s to some set X of vertices on the left, and edges connecting t to some set Y of vertices on the right. There are no edges from $\overline{X}$ to $\overline{Y}$, since otherwise there would be a path from s to t. Therefore, $S = X \cup Y$ is a vertex cover of the bipartite graph.

This shows that there is a vertex cover of size equal to the minimum cut, which is also the size of the maximum flow, and therefore of the maximum matching. We already showed that any vertex cover has to be at least this large, so we're done.

4.17 Suppose a graph G has a vertex cover S of size k. If we remove an edge, then S is still a vertex cover. If we contract an edge (u, v), then at least one of u or v must have been in S. If we include the new, combined vertex in the vertex cover, then it covers all the edges that u or v had before. In either case we get a vertex cover S' of the new graph G', where $|S'| \leq |S|$.

For the explicit algorithm, choose any edge (u, v). Branch into two subproblems. In one, include u in S and remove u and its edges from the graph. In the other, do the same with v. In each case, ask whether the remaining graph has a vertex cover of size $k - 1$. Working recursively produces a binary tree of depth k. At each leaf, we ask whether a graph has a vertex cover of size 0, which it does if and only if it has no edges. Each step requires $\mathrm{poly}(n)$ bookkeeping to remove a vertex from the graph, so the total running time is $2^k\,\mathrm{poly}(n)$.

4.19 If the witness is of length n, say, there are 2^n possible witnesses, and *a priori* searching through them all takes exponential time. But if the witness is of length $O(\log n)$, there are only $2^{O(\log n)} = \mathrm{poly}(n)$ possibilities, and we can check them all in polynomial time. To be more precise, if the witness is of length $A \log_2 n$ for some constant A, there are only $2^{A \log_2 n} = n^A$ possible witnesses.

4.20 If L is in NP, we can prove that v is in L^* by providing the breakpoints between the w_i, and giving a witness proving that each one is in L. Thus L^* is in NP.

If L is in P, we use dynamic programming. Let's define $f(w)$ and $f^*(w)$ as functions which return `true` or `false` depending on whether w is in L or L^* respectively. We already know that f is in P. We can define f^* as follows. If v is the empty word then $f^*(v) = \texttt{true}$. If $|v| = n$ for some $n > 0$, let v_i denote the word consisting of the first i symbols of v, and let $\overline{v_i}$ denote the rest of v. In other words, for any i we have $v = v_i \overline{v_i}$. Then we can write

$$f^*(v) = \bigvee_{i=1}^{n} \left(f(v_i) \wedge f^*(\overline{v_i})\right) .$$

In other words, $v \in L^*$ if there is some i such that $v_i \in L$, and the rest of v is in L^*. This gives a recursive algorithm, with the base case $f^*(v) = \texttt{true}$ if v is empty. If we memorize our previous results, we only need to call $f^*(\overline{v_i})$ once for each i, so there are only n different subproblems we need to solve. Thus dynamic programming gives a polynomial-time algorithm, and L^* is in P.

4.21 If q does not have a prime factor smaller than r, we can give q's prime factorization $q = p_1^{t_1} \ldots p_\ell^{t_\ell}$ as a witness, along with Pratt certificates, as described in Section 4.4.1, for the primality of $p_1, \ldots, p_\ell$. Thus SMALL FACTOR is in coNP.

If q does have a prime factor $p < r$, we can again give q's factorization as a witness, or (even easier) just give any divisor less than r. Thus SMALL FACTOR is also in NP.

4.22 This problem does not seem to be in NP. To prove that $\chi(G) = k$, it is not enough to show me a k-coloring. You also need to prove to me that no $(k-1)$-coloring exists. Thus while the claim that $\chi(G) \le k$ is in NP, the claim that it is exactly k seems to require a combination of an NP statement with a coNP one. We discuss problems like this, and with more complicated logical structures, in Section 6.1.1.

4.23 If x is the product of the first t primes, then x has 2^t distinct divisors (including 1 and itself). But since the tth prime is $O(t \ln t)$, we have $n = \log_2 x = O(t \log(t \log t)) = O(t \log t)$. Finally, if $n = O(t\,log\,t)$, then $t = \Omega(n/\log n)$.

4.24 Since the witness for p includes its tag and the tag of each q_i, we have the recurrence

$$T(p) = 1 + \sum_i T(q_i)$$

where the sum is over the distinct prime factors q_i of $p-1$ other than 2 (note that $p-1$ is even). We will prove by induction that $T(p) \le \log_2 p$ for all primes p. Assume this is true for primes smaller than p, and therefore for each q_i. Since p is odd, $p-1$ is even, and the product of the odd factors of $p-1$ is at most $(p-1)/2$. This gives

$$\begin{aligned} T(p) &\le 1 + \sum_i \log_2 q_i = 1 + \log_2 \prod_i q_i \\ &\le 1 + \log_2 \frac{p-1}{2} = \log_2 (p-1) < \log_2 p . \end{aligned}$$

So we need at most $O(\log p)$ tags.

Now, the size of p's tag is the number of bits of a primitive root $a < p$, which is at most $\log_2 p$, and the list of $p-1$'s prime factors q_i along with their exponents. In the worst case, the q_i are all distinct, and their total size in bits is

$$\sum_i \log_2 q_i = \log_2(p-1) \leq \log_2 p \,.$$

So the number of bits of each tag is $O(\log p)$, and the total size of the witness is $O(\log^2 p) = O(n^2)$ bits.

4.25 Peg Solitaire is in NP. Since each move removes a peg, the solution to a puzzle with n pegs involves less than n moves. This means that the witness has only polynomial length, and it is easily checked in polynomial time. For Sliding Blocks, on the other hand, the solution could involve an exponential number of moves, in which blocks are moved back and forth many times—indeed, check out "Junk's Hanoi" on page 346. There is no obvious way to compress such a solution, or prove its existence, with a witness of polynomial length.

Chapter 5

Who is the Hardest One of All? NP-Completeness

5.1 Let's start by looking at the 2-SAT and 3-SAT clauses corresponding to an AND:

$$y = x_1 \wedge x_2 \iff (x_1 \vee \overline{y}) \wedge (x_2 \vee \overline{y}) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y).$$

As in the reduction from 3-SAT to NAE-3-SAT, we can introduce a global dummy variable w, and think of a variable as true if its value is the opposite of w's. Equivalently, we focus our attention on the satisfying assignments where w is false. Then consider the NAE-3-SAT clauses

$$(x_1, \overline{y}, w) \wedge (x_2, \overline{y}, w) \wedge (\overline{x}_1, \overline{x}_2, y).$$

The first two clauses ensure that if either x_1 or x_2 is false (i.e., if either one is equal to w) then y must be false. The third clause then forces y to be true if x_1 and x_2 are both true.

Having expressed an AND gate, we can use De Morgan's law to express an OR gate. Finally, a NOT gate is easy: if $y = \overline{x}$, then include the clause (x, y, y), or if you prefer not to repeat variables, the clauses (x, y, w) and $(x, y, \overline{w})$.

Clearly we can carry out this reduction in polynomial time, and it produces a formula of polynomial size, since we simply replace each gate with a constant number of clauses.

5.2 1-IN-3 SAT is obviously in NP: the witness consists of a satisfying assignment, and we can check in polynomial time that exactly one literal in each clause is true.

To show that POSITIVE 1-IN-3 SAT is NP-complete, we reduce to it from GRAPH 3-COLORING. Given a graph G, for each vertex v, we define Boolean variables r_v, g_v and b_v corresponding to its three possible colors, where r_v is true if v is colored red and so on. Then we build a 1-IN-3 SAT formula ϕ as follows. For each v, we include the clause (r_v, g_v, b_v), which requires v to have exactly one of these colors. In addition, for each edge (u, v), we include the clauses (r_u, r_v, r_{uv}), (g_u, g_v, g_{uv}), and (b_u, b_v, b_{uv}). These clauses force u and v to be given different colors, and we set r_{uv} true if neither u nor v is given the color r. Clearly this reduction can be carried out in polynomial time, and G is 3-colorable if and only if ϕ is satisfiable.

5.3 A MAJORITY-OF-3 SAT clause (x, y, z) is equivalent to the three 2-SAT clauses $(x \vee y)$, $(y \vee z)$, and $(x \vee z)$. Thus it reduces to 2-SAT and is in P.

5.4 Let's call a Horn clause one with a single positive variable x_i. As we stated in the problem, a Horn clause is equivalent to an implication: x_i is forced to be true if all the other variables in the clause are true.

We start by setting all the variables false, and we will only set a variable true when we have to. This process begins when we satisfy the unit Horn clauses (x_i), which simply demand that x_i be true. We now iterate, checking each Horn clause at each step, and setting its positive variable true if all its negated variables are true. We continue in this way until all Horn clauses are satisfied. Clearly this gives the smallest possible set of variables which must be true in any satisfying assignment.

Each of these "rounds" takes time linear in the total length of the formula. And, the number of rounds is at most the number of variables, since we set a variable true in each round, and never reset a variable false.

Now all we have to do is check whether the other clauses—that is, those where every variable is negated—are satisfied. If for any of these clauses, we were forced to set all its variables true in the process of satisfying the Horn clauses, the formula is unsatisfiable. Otherwise, we have found a satisfying assignment.

5.5 Given an instance of VERTEX COVER with a graph $G=(V,E)$, generate a 2-SAT formula with a Boolean variable v for each vertex $v \in V$. Then for each edge $(u,v) \in E$, include the clause $(u \vee v)$, corresponding to including at least one endpoint of that edge in the vertex cover. Then a vertex cover of size t or less exists if and only if we can satisfy the formula by setting t or fewer variables true. This reduces VERTEX COVER to THRESHOLD 2-SAT, and proves that THRESHOLD 2-SAT is NP-complete.

5.6 We show that CONSTRAINED GRAPH 3-COLORING is in P by reducing it to 2-SAT. For each vertex v, define Boolean variables r_v, g_v, b_v, where r_v is true if v is colored red and false otherwise, and so on. Constraints between neighboring vertices u, v can be expressed as 2-SAT clauses, which force them to be different colors:

$$(\overline{r}_u \vee \overline{r}_v) \wedge (\overline{g}_u \vee \overline{g}_v) \wedge (\overline{b}_u \vee \overline{b}_v).$$

Similarly, we can write the restriction that each vertex has a single color, but not its forbidden one, as a 2-SAT clause. For instance, if v cannot be red, we write

$$(g_v \vee b_v),$$

forcing at least one of g_v and b_v to be true. (We can ensure that exactly one of them is true by also including the clause $(\overline{g}_v \vee \overline{b}_v)$, but this isn't necessary.) Clearly this is a polynomial-time reduction, so CONSTRAINED GRAPH 3-COLORING is in P since 2-SAT is.

On the other hand, we can reduce GRAPH $(k-1)$-COLORING to CONSTRAINED GRAPH k-COLORING by forbidding the same color at every vertex. Since GRAPH k-COLORING is NP-complete for $k \geq 3$, this shows that CONSTRAINED GRAPH k-COLORING is NP-complete for $k \geq 4$.

5.7 This is a variant of PERFECT MATCHING, in which, rather than asking that the edges in the matching don't overlap, we ask that they be a distance at least 2 apart from each other. This constraint feels like INDEPENDENT SET, and indeed we will prove that CELLPHONE CAPACITY is NP-complete by reducing INDEPENDENT SET to it.

Given an instance (G,k) of INDEPENDENT SET where $G=(V,E)$, we construct an instance (G',k') of CELLPHONE CAPACITY as follows. We set $k'=k$. To construct G', for each vertex $v \in V$ we add a new vertex

v', and add an edge (v, v'). We now claim that (G, k) has an independent set of size k if and only if G' has a set of conversations of size k.

In one direction, if $S \subseteq V$ is independent, we can construct a set of conversations in G' of the same size by having v talk to v' for each $v \in S$.

Conversely, suppose that there is a set C of conversations in G'. If we choose one endpoint from each edge (v, w) in C, the resulting set is independent. If $v, w \in V$, i.e., the conversation is between two of the original vertices of G, we can choose either one. If $v \in V$ and $w = v'$, we choose v. This gives an independent set $S \subseteq V$ of size k, so G has an independent set of size k. Thus the reduction maps yes-instances to yes-instances and no-instances to no-instances, and the proof is complete.

Clearly we can carry out this reduction in polynomial time, since we are simply replacing each vertex with a gadget of constant size.

5.8 Suppose we have a MAX CUT instance with a graph $G = (V, E)$ and a threshold k. For each vertex $v \in V$, define a Boolean variable x_v which is true or false depending on which side of the cut v is on. Then define a 2-SAT formula in which, for each edge $(u, v) \in E$, we have a pair of clauses $(x_u \vee x_v)$ and $(\overline{x}_u \vee \overline{x}_v)$. Both these clauses are satisfied if u and v are on opposite sides of the cut, but exactly one of them is satisfied if u and v are on the same side. Therefore, set the threshold for the number of clauses we wish to satisfy to $|E| + k$.

5.9 For $k \geq 3$, reduce from GRAPH k-COLORING simply by making k disjoint copies of the graph. If G is k-colorable, then by permuting the colors we can make the coloring of these k copies balanced. Conversely, if the k copies are k-colorable, then so is G.

For $k = 2$, each connected component of the graph has two 2-colorings, and in each one the number of black and white vertices differ by some integer x_i. This turns BALANCED k-COLORING into INTEGER PARTITIONING. However, these x_i are only polynomial. So, we can solve this instance of INTEGER PARTITIONING using dynamic programming, as discussed in Section 4.2.3.

5.11 We will reduce NAE-3-SAT to HYPERGRAPH 2-COLORING. However, since we can't "negate" the color of a vertex, we include two vertices for every variable x, which we call v_x and $v_{\overline{x}}$. The idea is that x is true if v_x is black and $v_{\overline{x}}$ is white, and vice versa. Then our hyperedge will have an edge for each NAE-3-SAT clause, which includes v_x or $v_{\overline{x}}$ if the clause includes x or $\overline{x}$.

However, we need to make sure that v_x and $v_{\overline{x}}$ are forced to have opposite colors. There are many ways to do this. If vertices are allowed to appear twice in an edge, we can simply include the edge $\{v_x, v_x, v_{\overline{x}}\}$ in the hypergraph. If repeated vertices are not allowed, we can add three vertices s, t, u and the edge $\{s, t, u\}$. Then for each x, we include the edges

$$\{v_x, v_{\overline{x}}, s\}, \{v_x, v_{\overline{x}}, t\}, \{v_x, v_{\overline{x}}, u\}.$$

Since s, t, u cannot all be the same color, v_x and $v_{\overline{x}}$ are forced to have opposite colors, and we're done.

Clearly we can carry out this reduction in polynomial time, since we are simply replacing each clause with a gadget of constant size.

5.12 Clearly this problem is in NP. The witness is the coloring, and we can check it in polynomial time.

Reduce from GRAPH 3-COLORING to HYPERGRAPH RAINBOW 3-COLORING as follows: for each edge (u, v) in the graph G, define a hyperedge consisting of u, v, and an additional vertex $w_{(u,v)}$. Call the resulting 3-uniform hypergraph H. If G is 3-colorable then H is rainbow colorable, since we can give $w_{(u,v)}$ whichever color u and v don't use. Conversely, if H is rainbow colorable then each hyperedge $(u, v, w_{(u,v)})$ forces u and v to have different colors, so G is 3-colorable. This reduction can clearly be carried out in polynomial time, so HYPERGRAPH RAINBOW 3-COLORING is NP-complete.

We can reduce from HYPERGRAPH RAINBOW 3-COLORING to HYPERGRAPH RAINBOW k-COLORING for any $k > 3$ by adding $k-3$ dummy vertices, and adding these to every hyperedge of H, giving a k-uniform hypergraph H'. Then H is rainbow colorable if and only if H' is. Since we already know that the case $k=3$ is NP-complete, this proves NP-completeness for any $k \geq 3$.

5.13 WHEEL 5-COLORING is clearly in NP, since we can check a coloring to see if it is valid in polynomial time.

We know that GRAPH k-COLORING is NP-complete for any $k \geq 3$. We will reduce GRAPH 5-COLORING to WHEEL 5-COLORING. Our reduction replaces each edge (u, v) of G with a gadget consisting of a path of length 3 between u and v. You can check that this gadget is wheel-5-colorable if and only if u and v have different colors, so it simulates an edge in traditional GRAPH 5-COLORING. Replacing each edge of G with this gadget gives a new graph G', such that G' is wheel-5-colorable if and only if G is 5-colorable. We can clearly carry out this reduction in polynomial time, and our proof is complete.

5.14 We claim that if we convert a 3-SAT formula ϕ with n variables and m clauses to a graph G using the gadgets in Figure 5.14, then G has an independent set S of size $n+m$ if and only if ϕ is satisfiable.

In one direction, suppose ϕ is satisfiable. Choose a satisfying assignment, and in each variable gadget include the vertex corresponding to the opposite of its truth value in S. Now, in each triangle, if the corresponding clause is satisfied, then at least one of its vertices v is connected to a variable vertex $w \notin S$, corresponding to one of the true literals in that clause. We then include v in S. This gives us one vertex for each variable and one for each clause, for a total of $n+m$.

Conversely, suppose that G has an independent set S. In each variable gadget and each clause gadget, at most one vertex can be in S, so if $|S| = n+m$, exactly one vertex in each gadget is in S. Assign each variable a truth value corresponding to which vertex in its gadget is not in S. Then for each clause gadget, at least one of its vertices is connected to the unoccupied vertex of its variable gadget, so the clause is satisfied by its truth value. Thus this is a satisfying assignment, and ϕ is satisfiable.

5.15 A covering of G with k cliques is simply a k-coloring of its complement $\overline{G}$. Therefore, CLIQUE COVER reduces directly to GRAPH COLORING and vice versa.

5.16 To show that it is in NP, a witness would consist of a k-coloring and a k-clique, for some k. The k-coloring shows that $\chi(G) \leq k$, and the k-clique shows that $\omega(G) \geq k$. But since $\chi(G) \geq \omega(G)$, this shows that $\chi(G) = \omega(G)$.

To show that is NP-complete, we reduce from GRAPH 3-COLORING. Given a graph G, we can check in polynomial time to see if any 4-cliques exist, and we know that G is not 3-colorable if they do. Therefore, we can assume without loss of generality that G has no 4-cliques. Create a graph G' by adding three vertices to G that form a triangle. Clearly G' is 3-colorable if and only if G is, and $\omega(G')=3$. Thus $\chi(G') = \omega(G')$ if and only if G is 3-colorable.

5.17 Clearly REALLY INDEPENDENT SET is in NP; the witness is the set S, and we can check in polynomial time that no two vertices in S have distance less than 3.

To show that it is NP-complete, we reduce from INDEPENDENT SET. Given a graph $G=(V,E)$, we will generate a graph $G'=(V',E')$ such that, for any $k>1$, G' has a really independent set S of size k if and only if G has an independent set S' of size k. The vertices of G' consist of a vertex w_v for each vertex $v\in V$, and a vertex w_e for each edge $e\in E$. The edges E' of G' are defined as follows: for each $e=(u,v)\in E$, we include (w_e,w_u) and (w_e,w_v) in E'. In addition, for all $e,e'\in E$ we include $(w_e,w_{e'})$ in E', so the set $\{w_e : e\in E\}$ forms a clique.

With this construction, the shortest-path distances in G' are as follows:

$$\text{for } u,v\in V,\quad d(w_u,w_v)=\begin{cases} 2 & (u,v)\in E \\ 3 & (u,v)\notin E \end{cases}$$

$$\text{for } v\in V,\, e\in E,\quad d(w_v,w_e)=\begin{cases} 1 & v\in e \\ 2 & v\notin e \end{cases}$$

$$\text{for } e,e'\in E,\quad d(w_e,w_{e'})=1$$

It follows that if G' has a really independent set S' of size k where $k>1$, then $S'=\{w_v : v\in S\}$ where $S\subseteq V$ is an independent set of size k in G. Conversely, if S is independent, then $S'=\{w_v : v\in S\}$ is really independent. Thus (G,k) is a yes-instance of INDEPENDENT SET if and only if (G',k) is a yes-instance of REALLY INDEPENDENT SET.

Finally, we note that this reduction can be carried out in polynomial time. Thus REALLY INDEPENDENT SET is NP-complete.

5.20 Let T be a set of ℓ rectangles where the ith one has height 1 and width x_i. Let R be a rectangle of height 2 and width $W/2$, where $W=\sum_{i=1}^{\ell} x_i$. Such an instance of RECTANGLE TILING is just INTEGER PARTITIONING in disguise, since we need to divide T into two subsets each of which has total width $W/2$. (We assume here that $x_i>2$ for all i so that none of the rectangles can fit vertically inside R; if you like, just double the widths so that this is true.) This provides a reduction from INTEGER PARTITIONING to RECTANGLE TILING, and proves that the latter is NP-complete.

5.23 Let us assume that (x,y) is an integer solution of (5.15). Then any common divisor of a and b also divides $ax+by=c$, and this is true in particular for $\gcd(a,b)$. For the reverse direction assume that $\gcd(a,b)$ divides c, i.e., there exists an integer $d>0$ such that $d\gcd(a,b)=c$. We know from the extended Euclidean algorithm that we can find integers x' and y' such that

$$ax'+by'=\gcd(a,b).$$

Multiplication of this equation with d then reveals $x=x'd$ and $y=y'd$ as solutions of (5.15).

Now if (x,y) is an integer solution of (5.15), so are

$$\left(x-\frac{kb}{\gcd(a,b)},\, y+\frac{ka}{\gcd(a,b)}\right)\qquad \text{for } k\in\mathbb{Z}.$$

Hence we have a one parameter set of solutions, and this set can easily and in polynomial time be searched for a solution $x>0$ and $y>0$.

5.24 According to the Chinese Remainder Theorem, each θ_i is less than the product of the moduli 2^m, q_j^m for $j \neq i$, and q_i. This is $q_i(2/q_i)^m K$, which if $q_1 < \cdots < q_n$ is at most $q_1(2/q_1)^m K$. Thus $2H = 2\sum_i \theta_i < 2nq_1(2/q_1)^m K$. If $m \geq 2$, then $q_1 > 8n$ is enough to ensure that $2H < K$. Thus we can use the tth prime for, say, $8n < t \leq 9n$, and we have $q_n = O(n \log n)$.

Since $\theta_i \equiv 0 \bmod q_j$ for $j \neq i$, we have $H \equiv \theta_i \not\equiv 0 \bmod q_i$, so H is not a multiple of q_i for any i. But, since $H^2 - X^2 \equiv 0 \bmod K$, we know that $(H+X)(H-X)$ is a multiple of q_i^m for all i. If both $H+X$ and $H-X$ were multiples of q_i, then their sum $2H$ would be, and so would H since q_i is odd. Thus one of $H+X$ or $H-X$ is mutually prime to q_i, and the other must be a multiple of q_i^m. Now we can define

$$\sigma_i = \begin{cases} +1 & \text{if } H-X \text{ is a multiple of } q_i^m \text{, i.e., } H \equiv X \bmod q_i^m \\ -1 & \text{if } H+X \text{ is a multiple of } q_i^m \text{, i.e., } H \equiv -X \bmod q_i^m . \end{cases}$$

Now let $X' = \sum_i \sigma_i \theta_i$. Since $\theta_i \equiv 0 \bmod q_j^m$ for $i \neq j$, for all i we have

$$H \equiv \theta_i \equiv \sigma_i X \equiv \sigma_i X' \mod q_i^m ,$$

and therefore $X \equiv X' \bmod K$. But $|X - X'| \leq 2H$, and if $2H < K$ this implies that $X = X'$.

We can reduce Subset Sum to the case with one odd weight by doubling t and each weight x_i, and adding $x_{n+1} = 1$. Thus we can assume without loss of generality that $S = 2t - \sum_i x_i$ is odd.

Now, if $X \equiv s \bmod 2^m$, $S+X$ is even, so $S^2 - X^2 = (S-X)(S+X) \equiv 0 \bmod 2^{m+1}$. Conversely, if $S^2 - X^2 \equiv 0 \bmod 2^{m+1}$, then $S-X$ and $S+X$ must be divisible by powers of 2 that multiply to give 2^{m+1}. But if both of them were multiples of 4, then their sum $2S$ would be, and S would be even. Thus one of them is a multiple of 2 but not of 4, and the other is zero mod 2^m, giving $X \equiv \pm S \bmod 2^m$.

5.25 Our choice of m implies $|S| < 2^m < q_i^m$, since $q_i > 2$ for all i. But $\theta_j \geq q_i^m$ for all $i \neq j$, since θ_j is a nonzero multiple of q_i^m. This easily implies that $|S| < \sum_i \theta_i = H$.

We set $\lambda_2 = -1$ and, say, $\lambda_1 = (K+1)^2$. Using $S^2 < H^2$ gives

$$2^{m+1} K Y > \big((K+1)^2 2^{m+1} - K\big)(H^2 - X^2),$$

so that $X \leq H$ implies $Y \geq 0$. Conversely, we have

$$2^{m+1} K Y \leq (K+1)^2 2^{m+1} (H^2 - X^2) + K X^2 .$$

If $Y \geq 0$, this implies

$$X \leq \sqrt{\frac{(K+1)^2 2^{m+1}}{(K+1)^2 2^{m+1} - K}} H \leq \left(1 + \frac{K}{(K+1)^2 2^{m+1}}\right) H \leq \left(1 + \frac{1}{K 2^{m+1}}\right) H \leq H + \frac{1}{2^{m+2}} ,$$

where we used the inequality $\sqrt{1/(1-z)} \leq 1 + z$ for $z \leq 1/2$ and $2H < K$. But since X and H are integers, this implies $X \leq H$.

5.26 Since the q_i are mutually prime, $\gcd(q_i, Q_i) = 1$. Thus we can use the extended Euclidean algorithm to find a P_i such that

$$P_i Q_i \equiv 1 \bmod q_i .$$

The x is then given by

$$x = (r_1 P_1 Q_1 + \cdots + r_\ell P_\ell Q_\ell) \bmod Q\,,$$

Since q_i divides every term on the right-hand side except $r_i P_i Q_i$, and $P_i Q_i \equiv 1 \bmod q_i$, we have $x \equiv r_i \bmod q_i$ for all i.

5.27 In the reduction from MAX BIPARTITE MATCHING to MAX FLOW, a partial matching M corresponds to a flow f where $f(e) = 1$ for each edge included in M, along with $f(e) = 1$ from s to the left endpoints of these edges and from their right endpoints to t, and $f(e) = 0$ for all other edges. According to the definition of the residual graph G_f, the edges in M become reverse edges with capacity 1. The value of the flow is $|M|$, the number of edges in M.

An augmenting path in G_f starts at s and zig-zags back and forth between the left and right vertices, alternating between forward and reverse edges, starting and ending with forward edges, and then arriving at t. Adding a unit of flow along this path adds the forward edges to M, and removes the reverse edges. The vertices immediately after s and immediately before t in this path are not covered by M. Thus if we remove the first edge out of s and the last edge into t, we are left with an alternating path in the bipartite graph. By adding 1 to the value of the flow, we increase the size of the matching to $|M| + 1$.

5.29 There are many ways to do this: we will reduce to REACHABILITY. Let S denote the alphabet of local states, such as $\{0,1\}$ in the example (5.14). If f depends only on nearest neighbors, and if the string a whose predecessor we are looking for is of length n, define a graph G with $|S|^2(n+1)$ vertices. Each vertex is labeled (s_1, s_2, x) where $s_1, s_2 \in S$ and $0 \le x \le n$. This vertex represents a guess that, in the predecessor state a', we have $a'_x = s_1$ and $a'_{x+1} = s_2$.

We draw an edge from $(s_1, s_2, x-1)$ to (s_2, s_3, x) if $f(s_1, s_2, s_3) = a_x$: in other words, if the pairs (s_1, s_2) and (s_2, s_3) overlap in a consistent way, and if the resulting three-site neighborhood (s_1, s_2, s_3) would give rise to a_x on the next step. Then a has a predecessor if and only if there is a path from $(s_1, s_2, 0)$ to (s_3, s_4, n) for some s_1, s_2, s_3, s_4.

If f depends on sites a radius r on either side rather than just on nearest neighbors, G will have $|S|^{2r}(n+1)$ vertices. Thus the problem gets exponentially harder as the radius increases.

For a dynamic programming algorithm, we can guess one site of the predecessor at a time, either starting from one end or cutting the string in the middle. Like many one-dimensional problems, either approach leads to a polynomial-time algorithm, since the number of subproblems increases only polynomially as a function of the length of the string.

5.30 SPIN GLASS is in NP, since the witness consists of the spins $\{s_i\}$. We can compute the energy in polynomial time, and check that it is less than or equal to E.

We reduce from unweighted MAX CUT, simply by setting $J_{ij} = -1$ whenever (i,j) are adjacent and $J_{ij} = 0$ otherwise. Then $E = \sum_{(i,j)} s_i s_j$ is the number of edges whose endpoints have equal spins, minus the number of edges whose endpoints have opposite spins. If we assign a spin $s_i = +1$ to all the vertices on one side of a cut, and $s_i = -1$ to the vertices on the other side, then E is $m - 2k$ where m is the total number of edges and k is the weight of the cut. Thus asking whether there is a cut of weight k or greater is the same as asking whether there is a state of energy $m - 2k$ or less.

5.32 Suppose we have an oracle, or subroutine, for INDEPENDENT SET CHANGE. We can find a maximal independent set in a graph G with n vertices by adding one edge at a time, calling this oracle $O(n^2)$ times.

The total time it takes to do this is polynomial. If we can find a maximal independent set, we can certainly check to see whether its size is at least k, so the decision problem INDEPENDENT SET is also in P. Since INDEPENDENT SET is NP-complete, this would imply that $\mathsf{P} = \mathsf{NP}$.

5.33 This one is easy: reduce from $(k/2+1)$-SAT by adding $k/2-1$ dummy variables to each clause.

Chapter 6

The Deep Question: P vs. NP

6.4 Suppose a problem A is in $\mathsf{NTIME}\big(f_1(g(n))\big)$. If we pad the input out to length $n' = g(n)$ by adding $g(n) - n$ zeros, the resulting problem is in $\mathsf{NTIME}(f_1(n'))$ and therefore in $\mathsf{TIME}(f_2(n'))$. But as a function of the original input length n, this is $\mathsf{TIME}\big(f_2(g(n))\big)$.

If $\mathsf{EXP} = \mathsf{NEXP}$, then a given problem in $\mathsf{NTIME}(2^n)$ is in $\mathsf{TIME}(2^{n^b})$ for some constant b. Then we can take $f_1(n) = 2^n$, $f_2(n) = 2^{n^b}$, and $g(n) = 2^{O(n^c)}$. Then $f_1(g(n)) = 2^{2^{O(n^c)}}$, and $f_2(g(n)) = 2^{\left(2^{O(n^c)}\right)^b} = 2^{2^{O(n^c)}}$.

6.5 Consider a problem A in NZOWIE. There is some constant k such that $A \in \mathsf{NTIME}(2 \uparrow^k n)$. But since $\mathsf{NTIME}(f(n)) \subset \mathsf{TIME}(2^{O(f(n))})$ for any $f(n)$, A can be solved deterministically in time

$$2^{O(2 \uparrow^k n)} .$$

This is essentially $2 \uparrow^{k+1} n$, except that the O could hide a large constant. This is overkill, but we can certainly handle this by exponentiating one more time, getting $A \in \mathsf{TIME}(2 \uparrow^{k+2} n)$ and therefore $A \in \mathsf{ZOWIE}$. Thus $\mathsf{NZOWIE} \subseteq \mathsf{ZOWIE}$, and since $\mathsf{ZOWIE} \subseteq \mathsf{NZOWIE}$ trivially we have $\mathsf{ZOWIE} = \mathsf{NZOWIE}$.

6.6 Each step of `MS` reduces an instance with n variables to at most three instances, with $n-1$, $n-2$, and $n-3$ variables respectively. Thus $T(n)$ is at most the number of recursive calls it takes to solve each of these instances, which is $T(n-1) + T(n-2) + T(n-3)$.

Since this looks like the Fibonacci equation, we try an exponential solution of the form $T(n) = c^n$. Plugging this in gives

$$c^n = c^{n-1} + c^{n-2} + c^{n-3} ,$$

and dividing both sides by c^{n-3} gives the cubic equation $c^3 - c^2 - c - 1 = 0$. Using the same kind of inductive proof as in Problem 2.4, we get that $T(n) = O(c^n)$ where c is the largest root of this equation, and where the constant in the O is determined by the initial values $T(0)$, $T(1)$, and $T(2)$. Since each recursive call takes polynomial time, either to compute the new formula ϕ or to call `2-SAT`, the total running time is $T(n)\,\mathrm{poly}(n) = c^n\,\mathrm{poly}(n)$.

For larger k, we have

$$c^n = c^{n-1} + c^{n-2} + \cdots + c^{n-k} ,$$

or, dividing both sides by c^{n-k},

$$c^k - c^{k-1} - c^{k-2} - \cdots - 1 = 0 .$$

If we multiply this by $c-1$, the terms between c^{k-1} and c cancel, giving

$$c^{k+1}-2c^k+1=c^k(c-2)+1=0\,.$$

This equation has all the roots we had before, plus $c=1$, so its largest root is still the c we're looking for.

6.9 Let $\chi_\Pi(y)=\texttt{true}$ if (Π,y) is a yes-instance of Π and $\texttt{false}$ if it is a no-instance. Similarly, let $\chi_{22}(y)=\texttt{true}$ or $\texttt{false}$ if (Π,y) is a yes- or no-instance of NCATCH22. By construction, $\chi_{22}(y)=\chi_\Pi(y+1)$ for all $y<2^b$, and $\chi_{22}(2^b)\neq\chi_\Pi(0)$. If $\chi_\Pi(y)$ is the same for all y, then $\chi_\Pi(2^b)\neq\chi_{22}(2^b)$. Otherwise, $\chi_\Pi(y)\neq\chi_{22}(y)$ for the smallest y such that $\chi_\Pi(y)\neq\chi_\Pi(y+1)$.

If $y<2^b$, the number of steps it takes to run NCATCH22 is $s(f\left(|(\Pi,y)|\right)=s(f(n))$, not counting the time it takes to "wind the clock." If $y=2^b$, the number of steps it takes to check all possible witnesses z of length at most $f\left(|(\Pi,0)|\right)$ is

$$2\cdot 2^{f(|(\Pi,0)|)}f\left(|(\Pi,0)|\right)=2^{f(|\Pi|+1)}f(|\Pi|+1)\,.$$

We set b large enough so that

$$2\cdot 2^{f(|\Pi|+1)}f(|\Pi|+1)\le f(b+1)\le f\left(|(\Pi,2^b)|\right)=f(n)\,.$$

For instance, if $f(n)=n^a$ for $a>1$, then $b=3^{|\Pi|}$ suffices for all $|\Pi|>10$. In that case, NCATCH22 is in $\mathsf{NTIME}\big(s(f(n))\big)\subseteq\mathsf{NTIME}(n^d)$ for any $d>a$, but it is outside $\mathsf{NTIME}(n^c)$ for any $c<a$.

6.10 Let's start with the largest independent set. Since its size is at most n on a graph with n vertices, we don't even need binary search: we ask the oracle whether there is an independent set of size 1, then size 2, and so on, and stop at the largest k for which she says "yes."

We then choose a vertex v, define a graph G' by removing it, and ask her whether G' has an independent set of size k. If she says "yes," we remove it permanently from G. If she says "no," we add it to the independent set S, remove it and its neighbors from G, and decrement k. We continue in this way until every vertex has either been removed from the graph or placed in S. The total number of questions we ask the oracle is at most n, so this is in P^{NP}.

If we can find the largest independent set S in G, then we can also find the smallest vertex cover, since this is just $\overline{S}$. Similarly, the largest clique in G is the largest independent set in $\overline{G}$.

6.12 If an OR gate has fan-in k, then the probability that none of its inputs are set to $\texttt{true}$ is $((1+p)/2)^k<(2/3)^k$. This is $o(n^{-2a})$ if $k=r\log n$ where $r>2a/\log(3/2)$. Using the union bound over the $O(n^a)$ OR gates (see Appendix A.3), with probability $1-o(n^{-a})$ no OR with $k>r\log n$ survives.

We now assume that all surviving OR gates have $k\le r\log n$. In that case, the probability that a given OR has c or more variables are unset is at most

$$\binom{k}{c}p^c\le\left(\frac{\mathrm{e}kp}{c}\right)^c,$$

where we used the union bound over the $\binom{k}{c}$ possible subsets of c variables, and the upper bound (A.10) on the binomial. Since $p=n^{-1/2}$ and $k=O(\log n)$, this probability is $O(n^{-c/2}\log^c n)$, so setting $c>4a$ makes this probability $o(n^{-2a})$. Thus the probability that a given OR gate survives with fan-in greater than k, or that it depends on more than c unset variables, is $o(n^{-2a})$. Again using the union bound, the probability that this holds for any of the $O(n^a)$ OR gates is $o(n^{-a})$.

Assuming that all surviving OR gates depend on at most c variables, in the second stage a given OR gate is set to `false` if all of its inputs are set to `false`. The probability of this is $((1-p)/2)^c > (1/3)^c$, a constant. If the H_i are disjoint then these events are independent, and the AND gate survives with probability less than $(1-(1/3)^c)^t$. If there are ℓ disjoint H_i, the same argument applies, and the probability of survival is $o(n^{-a})$ if $\ell > r \log n$ for a suitable constant r.

The base case of the induction is given by exactly the same argument we used to show that the OR gates have fan-in c after the first stage, except that `true` and `false` are switched.

For the inductive step, given the argument above we can assume that $\ell = O(\log n)$. Then $|H'| \le O(c \log n) = O(\log n)$, and the probability that c' or more variables in H' survive is at most $\binom{|H'|}{c'} p^{c'}$. This is again $o(n^{-a})$ if c' is a sufficiently large constant. So, we assume that at most c' variables in H' survive.

Now, if we were to set these c' variables, the AND would depend only on the OR gates not in I, and on the variables notin H'. But since I is the largest disjoint family, H' contains at least one variable from every OR, and each of them would then depend on at most $c-1$ variables. By induction, for each assignment of these c' variables, the AND would depend on at most h_{c-1} of the other variables. Summing over the $2^{c'}$ possible assignments, the AND depends on a total of at most h_c variables, where

$$h_c = c' + 2^{c'} h_{c-1} .$$

Each of the bounds we assumed on the gates' fan-ins and dependences is violated with probability $o(n^{-a})$, so this bound also holds with probability $1-o(n^{-a})$.

We can change an AND of ORs to an OR of ANDs—that is, change a CNF formula to a DNF formula—using the distributive law. If the formula has h variables where $h = h_c$, then there are just 3^h possible ANDs we can make out of them, one for each subset and choice of negation. Thus the circuit can be expressed as the OR of at most 3^h ANDs, a constant number.

6.13 We assume without loss of generality that the first and second layers consist of OR and AND gates respectively. Lemma 6.11 shows that after the restriction, a given AND gate on the second layer can be expressed, with probability $1-o(n^{-a})$, as the OR of a constant number of ANDs. Using the union bound, this is true of all $O(n^a)$ AND gates on that layer with probability $1-o(1)$. Then, since the second and third layer now both consist of OR gates, we can merge them into a single layer, giving a circuit of depth $d-1$ and width $O(n^a)$.

Since each variable is unset with probability $n^{-1/2}$, the Chernoff bound shows that the probability of unset variables is less than, say, half of its expectation $n^{1/2}$ is exponentially.

6.14 First note that if $\gamma_1(n) \le \gamma_2(n)$ and f is γ_1-complicated, then f is γ_2-complicated. Therefore, if f is γ-complicated where $\gamma(n) = n^{o(1)}$, then f is n^ε-complicated for any constant $\varepsilon > 0$.

Now suppose that f is in AC^0, and can be computed by a function of depth d. By applying Problem 6.13 d times, randomly restricting all but $\Omega(n^{1/2})$ variables each time, we get a circuit of depth zero—that is, a constant function—even though $\Omega(n^{1/2^d})$ variables remain unset. Therefore f cannot be n^ε-complicated for any $\varepsilon < 1/2^d$, giving a contradiction.

PARITY isn't a constant until no unset variables remain. If $d = c \log_2 \log_2 n$, then the number of unset variables remaining after reducing the depth to zero is roughly $n^{1/2^d} = n^{1/(\log_2 n)^c} = 2^{(\log_2)^{1-c}}$. If $c < 1$, this is $\omega(1)$, so the function cannot be constant. Thus any circuit for PARITY has to have depth at least $\log_2 \log_2 n$. (To make this rigorous, we need to check the probability that we can reduce the size of the circuit, and that $\Omega(n^{1/2^d})$ variables remain after applying Problem 6.13 d times where d is not a constant.)

In fact, a stronger version of the Switching Lemma due to Håstad can be used to show that Parity requires circuits of depth $\Omega(\log n / \log\log n)$.

6.15 Assuming that $\gamma(n)$ itself can be computed efficiently, we can check all $2^{O(n)}$ restrictions outside all $\binom{n}{\gamma(n)}$ subsets S of size $\gamma(n)$, and all $2^{\gamma(n)}$ values of f for each one, to confirm that none of the restrictions of f to $\gamma(n)$ variables is constant. Since $2^{O(n)} = \mathrm{poly}(2^n)$, this is polynomial as a function of the size of the truth table, and for any $\gamma(n)$ it can be computed by a circuit of constant depth and $\mathrm{poly}(2^n)$ width.

6.16 Let $f(x)$ be a random function. The number of subsets S of any size, times the number of ways to set the variables outside S, is at most $3^n = 2^{O(n)}$. For each such restriction, the $2^{|S|} = 2^{\gamma(n)}$ possible values of f are independently random, so the probability that a given restriction of f is constant is $2 \times 2^{-2^{\gamma(n)}}$. If $\gamma(n) = A \log_2 n$ where $A > 1$, then the expected number of restrictions where f is constant is $2^{O(n)-n^A} = 2^{-n^A}$. This is exponentially small, and by the first moment method so is the probability that there are any such restrictions.

6.23 First we show that every subgraph $K \subseteq H$ has even degree. If K has nonzero degree—that is, if it is a subgraph of a Hamiltonian path p on $H \cup \overline{H}$—then it consists of one or more paths. Let q be the first path in K in some convenient order. Any path p such that $K \subset p$ passes through q by wiring it to the other vertices. But then we can define a partner p' by reversing this wiring, swapping q's endpoints, and passing through q in the other direction. This defines a perfect matching on K's neighborhood $\{p : K \subset p\}$, so K has even degree.

Now consider a Hamiltonian path p on $H \cup \overline{H}$. It has a total of n edges. Suppose that s of its edges are in H and $t = n - s$ are in $\overline{H}$. If $t > 0$, then p has 2^s subgraphs $K \subseteq H$, none of which is equal to p. This is odd if $s = 0$, and even otherwise. If $t = 0$ and $s = n$, then p has $2^n - 1$ subgraphs other than itself. Thus p's degree is odd if $s = 0$ or $t = 0$—that is, if its edges are either all in H or all in $\overline{H}$—and even otherwise.

6.24 We defined $p' = \phi(K, p)$ in the solution to the previous problem: reverse the order in which p passes through the first path q in K.

On the other side, let $K' = \psi(p, K) = (p \cap H) - K$. If p has edges in both H and $\overline{H}$, this gives a perfect matching on the set of $K \subseteq H$ such that $K \subset p$. If all of p's edges are in H, then $\phi(p, \emptyset) = p$, but since we demand that $K' \neq p$ we leave $\emptyset$ unmatched. Similarly, if all of p's edges are in $\overline{H}$, then $\phi(p, \emptyset) = \emptyset$, and $\emptyset$ is matched to itself.

Chapter 7

The Grand Unified Theory of Computation

7.1 Each program is just a finite string, which we can interpret as an integer in base b if our language uses an alphabet with b symbols. Thus the number of programs is countable.

Let's assume that the functions $f : \mathbb{N} \mapsto \mathbb{N}$ are countable. Then we can label the functions f_0, f_1, f_2 etc., and we can compile the complete list of functions:

$$\begin{array}{ccccc}
\mathbf{f_0(0)} & f_0(1) & f_0(2) & f_0(3) & \cdots \\
f_1(0) & \mathbf{f_1(1)} & f_1(2) & f_1(3) & \\
f_2(0) & f_2(1) & \mathbf{f_2(2)} & f_2(3) & \\
f_3(0) & f_3(1) & f_3(2) & \mathbf{f_3(3)} & \\
\vdots & & & & \ddots
\end{array}$$

Now consider the function $g(n) = f_n(n) + 1$. This function differs from all f_k in the list, hence the list cannot be complete, contrary to our assumption.

Alternately, we can simply note that functions $f : \mathbb{N} \to \{\texttt{true}, \texttt{false}\}$ are in one-to-one correspondence with subsets $S \subseteq \mathbb{N}$, namely $S = \{x : f(x) = \texttt{true}\}$.

7.2 If your program checks one odd number after the other, starting from 3 (or 2501 if you trust Euler), it should stop at 5777, for which the conjecture doesn't hold. Another counterexample is 5993. Both of these were discovered by Moritz Stern (1807–1894) and, curiously enough, they are the only counterexamples known to this day. A refurbished conjecture states that only a finite number of odd numbers cannot be written as $p + 2a^2$.

7.3 Suppose we have an instance (Π, x) of HALTING. We will convert it to a program Φ which is an instance of each of these problems. Note that in each case, x is "hard-coded" into Φ.

We can solve the first 4 cases with the same reduction. Given (Π, x), we write a program $\Phi(y)$ that ignores its input y, runs $\Pi(x)$, and returns `true` if $\Pi(x)$ halts. Thus if $\Pi(x)$ halts,

1. $\Phi(0)$ halts,

2. $\Phi(y)$ halts for all y,

3. $\Phi(y) = \texttt{true}$ for all y, and
4. the set of y such that $\Phi(y)$ halts is $\mathbb{N}$, which is infinite.

If $\Pi(x)$ doesn't halt then $\Phi(y)$ is undefined for all y, so none of (1), (2), or (3) hold. The set of y such that $\Phi(y)$ halts is $\emptyset$, which is finite. In case (4), Φ is a yes-instance if and only if (Π, x) is a no-instance.

For case (5), given (Π, x) we write a program $\Phi(y)$ that runs $\Pi(x)$ and then returns y if $\Pi(x)$ halts. Then we either have $\Phi(y) = y$ for all y, or $\Phi(y)$ is undefined for all y.

For case (6), let's start by assuming that Ψ halts on at least one input. Then given (Π, x), we write a program $\Phi(y)$ that runs $\Pi(x)$, and then runs $\Psi(y)$ if $\Pi(x)$ halts. Then Φ and Ψ compute the same partial function if $\Pi(x)$ halts. Otherwise, $\Phi(y)$ is always undefined, and is not equivalent to Ψ.

If Ψ never halts so that its partial function is always undefined then we flip yes- and no-instances, and write $\Phi(y)$ so that it always returns $\texttt{true}$ if $\Pi(x)$ halts. In this case, Φ and Ψ are equivalent if $\Pi(x)$ doesn't halt.

7.4 Suppose that P is false for the program that never halts, i.e., the partial function which is undefined for all inputs. Then given an instance (Π, x) of HALTING, write a program $\Phi(y)$ which runs $\Pi(x)$ and then, if $\Pi(x)$ halts, runs $\Pi_1(y)$ and returns its output. Thus if $\Pi(x)$ halts, Φ computes the same partial function as Π_1. Thus P is true of Φ if and only if $\Pi(x)$ halts.

Alternately, if P is true for the program that never halts, we do the same thing with Π_2. Then P is true of Φ if and only if $\Pi(x)$ doesn't halt.

7.8 If S is finite then it is decidable, since given x we can check whether $x = s$ for each $s \in S$ in finite time. So we can assume that S is infinite.

Now suppose we want to know if a given x is in S. We run the enumerating program Π. One of two things happens in finite time: either Π prints x, or it skips x and prints some $x' > x$ (we know there is such an x' since S is infinite). In either case, we learn whether or not $x \in S$ in finite time.

7.14 First, note that by multiplying with $\texttt{leq}(x, y)$ we can condition on whether $x \le y$ or not. Then define $f(x) = h(x, t(x))$ where

$$h(x,0) = 0$$
$$h(x,y) = \begin{cases} y & \text{if } g(y) \le x \\ h(x, y-1) & \text{if } g(y) > x \,. \end{cases}$$

For $\lfloor \log_2 x \rfloor$, take the special case $t(x) = x$ and $g(y) = 2^y$.

7.15 We can write $\texttt{pair}(x, 0)$ in primitive recursive fashion as

$$\texttt{pair}(x,0) = 0$$
$$\texttt{pair}(x+1,0) = \texttt{pair}(x,0) + x + 1\,,$$

and then

$$\texttt{pair}(x,y) = \texttt{pair}(x+y,0) + y\,.$$

If you prefer an explicit formula, we have

$$\texttt{pair}(x,0)=\frac{x(x+1)}{2} \text{ and } \texttt{pair}(x,y)=\frac{(x+y)(x+y+1)}{2}+y\,.$$

To find $\texttt{left}(p)=x$, we use Problem 7.14 to find the right "block" of x-values, where each new block starts with $y=0$:

$$\texttt{block}(p)=\max\{x : x\le p \text{ and } \texttt{pair}(x,0)\le p\}$$

Then

$$\begin{aligned}\texttt{left}(p)&=\texttt{block}(p)-(p-\texttt{pair}(\texttt{block}(p),0))\\ \texttt{right}(p)&=p-\texttt{pair}(\texttt{block}(p),0)\,.\end{aligned}$$

7.17 The inequality (7.30) holds for $n=2$ since $2^y\ge 2y+2$ for all $y\ge 3$. We then proceed by induction on n. For each $n\ge 3$, we use induction on y. For the base case $y=3$, (7.30) holds with equality since

$$G_n(0)=1\,,\ G_n(1)=2\,,\ G_n(2)=4\,,\ G_n(3)=G_{n-1}(G_n(2))=G_{n-1}(4)\,.$$

Then, for all $y\ge 4$,

$$G_n(y)=G_{n-1}(G_n(y-1))\ge G_{n-1}(G_{n-1}(y))\ge G_{n-2}(G_{n-1}(y))=G_{n-1}(y+1)\,,$$

where we used the fact that $G_{n+1}(y)\ge G_n(y)$ for all $n\ge 2$ and $G_n(y+1)>G_n(y)$ for all n. (These facts are easily established by induction.)

Now suppose that $f(y)<G_n(y)$ and $g(y)<G_m(y)$. Then ignoring inputs and function values less than 3,

$$\begin{aligned}f(g(y))&<G_n(g(y))<G_n(G_m(y))\\ &<G_{\max(n,m)}(G_{\max(n,m)+1}(y))\\ &=G_{\max(n,m)+1}(y+1)\\ &\le G_{\max(n,m)+2}(y)\,.\end{aligned}$$

so their composition is majorized by $G_k(y)$ with $k=\max(n,m)+2$.

For the function $h(y)=g^y(f(0))$ defined by iteration, there is some z such that $G_{m+1}(z)=G_m^z(1)>f(0)$. Then

$$h(y)=g^y(f(0))<G_m^y(G_m^z(1))=G_m^{y+z}(1)=G_{m+1}(y+z)\le G_{m+z+1}(y)\,,$$

where in the last inequality we used (7.30) z times. Thus $h(y)$ is majorized by $G_\ell(y)$ where $\ell=m+z+1$.

By induction, every primitive recursive function is majorized by $G_n(y)$ for some n, and we're done.

7.21 Below is a version of **Q** in C++. Because of C++'s input conventions, we found it more convenient to represent Π as an array of strings `x` rather than a single string. Note the acrobatics we have to do to put backslashes in front of quotes and other backslashes. If we call this `q.C`, then we can feed it to itself in Unix with `q < q.C > qq.C`. Then if we compile and run `qq`, it prints out its own source code `qq.C`.

```
#include <iostream>
#include <string>
using namespace std;
void esc(string s) {
  for (int i=0; i<s.length(); i++) {
    char c=s.at(i);
    if (c==’\\’ || c==’\"’ || c==’\”) cout « "\\";
    cout « c;
  }
}
int main(void) {
  string x[100];
  int i;
  i=0; while (!cin.eof()) getline(cin,x[i++]);
  i=0; while (i<13) cout « x[i++] « endl;
  i=0; while (!x[i].empty()) {
    cout « "x[" « i « "]=\"";
    esc(x[i++]);
    cout « "\";" « endl;
  }
  i=14; while (!x[i].empty()) cout « x[i++] « endl;
}
```

7.22 We have

$$\begin{aligned}\mathbf{add}\,\mathbf{c}_0\,\mathbf{c}_j &= \big(\lambda nmfx\,.\,nf(mfx)\big)(\lambda gy\,.\,y)(\lambda hz\,.\,h^j z)\\ &= \big(\lambda mfx\,.(\lambda gy\,.\,y)f(mfx)\big)(\lambda hz\,.\,h^j z)\\ &= (\lambda mfx\,.\,mfx)(\lambda hz\,.\,h^j z)\\ &= \lambda fx\,.(\lambda hz\,.\,h^j z)fx\\ &= \lambda fx\,.\,f^j x\\ &= \mathbf{c}_j\,,\end{aligned}$$

and

$$\mathbf{add}'\,\mathbf{c}_0\,\mathbf{c}_j = (\lambda n\,.\,nS)(\lambda gy\,.\,y)\mathbf{c}_j = (\lambda gy\,.\,y)S\mathbf{c}_j = \mathbf{c}_j\,.$$

Then for the induction step,

$$\mathbf{add}(\mathbf{succ}\,\mathbf{c}_i)\mathbf{c}_j = \lambda fx\,.\,\mathbf{succ}\,\mathbf{c}_i f(\mathbf{c}_j fx) = \lambda fx\,.\,f(\mathbf{c}_i f(\mathbf{c}_j fx)),$$

and

$$\begin{aligned}\mathbf{succ}(\mathbf{add}\,\mathbf{c}_i\,\mathbf{c}_j) &= \big(\lambda nfx\,.\,f(nfx)\big)\big(\lambda gy\,.\,\mathbf{c}_i g(\mathbf{c}_j gy)\big)\\ &= \lambda fx\,.\,f\big((\lambda gy\,.\,\mathbf{c}_i g(\mathbf{c}_j gy))fx\big)\\ &= \lambda fx\,.\,f(\mathbf{c}_i f(\mathbf{c}_j fx)),\end{aligned}$$

while

$$\mathbf{add}'(\mathbf{succ}\,\mathbf{c}_i)\,\mathbf{c}_j = \mathbf{succ}\,\mathbf{c}_i\,\mathbf{succ}\,\mathbf{c}_j = \mathbf{succ}(\mathbf{c}_i\,\mathbf{succ}\,\mathbf{c}_j) = \mathbf{succ}(\mathbf{add}'\,\mathbf{c}_i\,\mathbf{c}_j)\,.$$

7.23 If we define

$$\Phi = \lambda p\,.\,\mathbf{pair}(\mathbf{second}\,p)(S(\mathbf{first}\,p))\,,$$

then the first component of $\Phi^n(0,0)$ is the predecessor of n, so

$$\mathbf{pred} = \lambda n\,.\,\mathbf{first}(n\Phi(\mathbf{pair}\,\mathbf{c}_0\,\mathbf{c}_0))\,.$$

7.24 We define a function which maps the pair (i,j) to $(i+1,ij)$:

$$\Phi = \lambda p\,.\,\mathbf{pair}(S(\mathbf{first}\,p))(\mathbf{mult}(\mathbf{first}\,p)(\mathbf{second}\,p))\,.$$

Then

$$\mathbf{fac} = \lambda n\,.\,\mathbf{second}(n\Phi(\mathbf{pair}\,\mathbf{c}_1\,\mathbf{c}_1))\,.$$

7.25 We have $h(x,0) = f(x)$ and $h(x,y+1) = g(x,y,h(x,y))$. Since g takes a triplet (x,y,h) as input, we define the following functions, analogous to the pair functions in Problem 7.23:

$$\mathbf{triplet} = \lambda xyzf\,.\,fxyz$$
$$\mathbf{1st} = \lambda p\,.\,p(\lambda xyz.x)\,,\ \mathbf{2nd} = \lambda p\,.\,p(\lambda xyz.y)\,,\ \mathbf{3rd} = \lambda p\,.\,p(\lambda xyz.z)\,.$$

We use the following function to "pack" g, so iterating it updates $h(x,y)$ while incrementing y and leaving x the same:

$$\tau = \lambda gt\,.\,\mathbf{triplet}(\mathbf{1st}\,t)(\mathbf{succ}(\mathbf{2nd}\,t))(g(\mathbf{1st}\,t)(\mathbf{2nd}\,t)(\mathbf{3rd}\,t))\,.$$

Then we have

$$h = \lambda xy\,.\,\mathbf{3rd}\big(y(\tau g)(\mathbf{triplet}\,x\,\mathbf{c}_0\,(fx))\big)\,.$$

By abstracting f and g, we get the primitive recursion operator which returns h:

$$\mathbf{pr} = \lambda fgxy\,.\,\mathbf{3rd}\big(y(\tau g)(\mathbf{triplet}\,x\,\mathbf{c}_0\,(fx))\big)\,.$$

Of course, what we're really doing here is replacing primitive recursion with iteration as in Problem 7.16. Handling iteration is quite easy, since we can write (7.29) as

$$h = \lambda y\,.\,y\,g(f\,\mathbf{c}_0)$$

or, if we want to write the iteration operator as a combinator that returns h,

$$\mathbf{iter} = \lambda fgy\,.\,y\,g(f\,\mathbf{c}_0)\,.$$

7.26 For Turing's combinator, we can write

$$\Theta = \eta\eta = \lambda y\,.\,y(\eta\eta y) = \lambda y\,.\,y(\Theta y)\,,$$

in which case $\Theta\mathbf{R} = \mathbf{R}(\Theta\mathbf{R})$. For Tromp's combinator, first write

$$\mathbf{T} = \mathbf{T}_1\mathbf{T}_2 = \lambda y\,.\,\mathbf{T}_2\,y\,\mathbf{T}_2\,.$$

Then since $\mathbf{T}_2\,y\,\mathbf{T}_2 = y(\mathbf{T}_2\,y\,\mathbf{T}_2)$, we have

$$\mathbf{T}\mathbf{R} = \mathbf{T}_2\,\mathbf{R}\mathbf{T}_2 = \mathbf{R}(\mathbf{T}_2\,\mathbf{R}\mathbf{T}_2) = \mathbf{R}(\mathbf{T}\mathbf{R})\,.$$

7.32 One solution, probably not the most efficient one, to the power of 2 problem is

$$\frac{13}{17} \quad \frac{17}{13} \quad \frac{10}{21} \quad \frac{14}{15} \quad \frac{1}{5} \quad \frac{1}{7} \quad \frac{3}{4} \quad \frac{13}{6} \quad \frac{15}{11} \quad \frac{11}{3}\,.$$

7.33 The FRACTRAN program

$$\frac{7\cdot 11}{2\cdot 3\cdot 5} \quad \frac{5}{11}$$

implements the mapping

$$2^a 3^b 5 \mapsto 2^{a-\min(a,b)} 3^{b-\min(a,b)} 7^{\min(a,b)} 5\,.$$

Note that in the output, at least one of the counters 2 or 3 contains a zero. To discriminate the three cases $a > b$, $a < b$ and $a = b$, we add three "exit"-fractions and two fractions that correct the decrement that comes with checking the counters:

$$\frac{7\cdot 11}{2\cdot 3\cdot 5} \quad \frac{13}{3\cdot 5} \quad \frac{17}{2\cdot 5} \quad \frac{19}{5} \quad \frac{5}{11} \quad \frac{3\cdot 29}{13} \quad \frac{2\cdot 23}{17}$$

Chapter 8

Memory, Paths, and Games

8.2 Read the input from left to right. Maintain a counter c with initial value zero, and increment or decrement it when reading '(' or ')' respectively. Reject if c ever becomes negative, and accept if $c = 0$ at the end of the input. We need to keep track of c and our current position on the input. Both of these are integers between 0 and n, so we just need $O(\log n)$ bits of memory.

8.3 Let us call say that each symbol has a *type*, either round or square, and say that each symbol is a left or right bracket regardless of type. Each left bracket a has a right bracket b which is its "partner," and our goal is to check that every left bracket's partner is of the same type. To find its partner we use a counter c as in question #1 above: we set $c = 0$ and then scan rightward from a, and a's partner is the symbol for which $c = -1$ for the first time.

One way to prove that this works is to realize that this set of strings can be generated by the following context-free grammar (see Problem 3.30):

$$S \to (S)S, [S]S, \varepsilon$$

where ε is the empty string. This means that w is a legal string if and only if it can be written $(w_1)w_2$ or $[w_1]w_2$ where w_1 and w_2 are legal strings. So if the string inside each matching pair of brackets is legal, the entire string is legal.

8.4 Problem 8.2 can be solved by reading the input from left to right, and incrementing or decrementing a counter when we read '(' or ')' respectively. Since the value of this counter never exceeds n, it can be stored in $O(\log n)$ bits of memory.

For Problem 8.3, let's say that two strings u, v are equivalent if, for all w, the concatenated string uw is properly nested if and only if vw is. The state of its memory is the only information the algorithm will ever have about the part of the input it has read so far, since it is not allowed to go back and read it again. So, if two words are not equivalent then reading them must put the memory into two different states, since otherwise the algorithm can't remember the difference between them. Since a memory with m bits can be in 2^m different states, the number of bits of memory we need is at least $\log_2$(# of equivalence classes).

Now, for each n, there are 2^n words of length n consisting of left parentheses and brackets. For $n = 3$, for instance, we have 8 strings `(((`, `(([`, `([(`, `([[`, ..., `[[[`. None of these are equivalent, since each one can be followed by its own mirror image but not by the mirror images of any of the others. The situation for palindromes is similar: given two distinct words of the same length, each one can be followed by its own mirror image but not by the other's. Thus there are at least 2^n equivalence classes of words of length n, and we need at least n bits of memory.

8.5 Let x_i and y_i denote the 2^i-bit of x and y respectively. If we start with the least-significant bits x_0 and y_0 and move to the left as we did in grade school, at each point we only need to keep track of two bits: r_j, the jth bit of the result, and c_j, the carry bit. Initially, we set $c_0 = 0$. Then for each j, we have $r_j = x_j \oplus y_j \oplus c_j$ where $\oplus$ denotes addition mod 2, and $c_{j+1} = 1$ if at least two of x_j, y_j, c_j are 1. When $j = i$, we return "yes" if $r_i = 1$. The only other thing we need to keep track of is j, and since this is an integer between 0 and n this takes $O(\log n)$ memory.

8.6 We can check a witness consisting of a series of moves. The witness checker can clearly verify that each move is legal given the current positions of the pebbles and read-only access to G. It takes $k \log n$ bits of memory to keep track of the position of the k pebbles, so the witness checker needs $O(k \log n)$ memory. If $k = O(1)$ then $k \log n = O(\log n)$ and the problem is in NL. If $k = \Theta(n)$, on the other hand, all we can say is that it is in NSPACE($n \log n$) ⊂ PSPACE.

Alternately, we can think of this as a Reachability problem on a larger graph G', where each vertex corresponds to a placement of the pebbles, and where two placements are adjacent if we can change one to the other by moving a single pebble. The number of different placements, and therefore the number of vertices of G', is $n' = \binom{n}{k} k! \le n^k$. If $k = O(1)$ then $n' = \mathrm{poly}(n)$, and G' is of polynomial size. We can compute the adjacency matrix of G' one bit at a time given read-only access to G—this is just checking that moves are legal—so this gives an L-reduction from the pebble problem to Reachability on a graph of polynomial size. If $k = \Theta(n)$, however, G' is exponentially large. In that case, this problem might be PSPACE-complete.

8.7 First we show that Strong Connectedness is in NL. If G has n vertices, we do a pair of nested **for** loops with u and v ranging from 1 to n. For each pair u, v we nondeterministically guess a path from u to v. Alternately, we read a witness which gives a path from each u to v. We only need to keep track of u, v, and the current vertex in our path, and all of these are $O(\log n)$-bit numbers.

Now we will show that Strong Connectedness is NL-complete by reducing Reachability to it. Given a directed graph G and two vertices s and t, we can generate a graph G' such that G' is strongly connected if and only if there is a path in G from s to t. We change G into G' by adding some additional edges: specifically, for all vertices v we add edges $u \to s$ and $t \to v$.

To prove that this works, suppose that G has a path from s to t. Then G' is strongly connected, since we can get from any vertex u to any other vertex v by going from u to s, along this path to t, and then from t to v. Conversely, suppose that G does not have a path from s to t. Adding these edges to G' cannot create such a path, since the $u \to s$ edges can only help us return to s, and the $t \to v$ edges are useless if we can't reach t in the first place. Thus G' does not have a path from s to t, so it is not strongly connected.

Finally, to show that this is a log-space reduction, we just need to show that given read-only access to G, s and t, we can answer the question whether there is an edge from u to v in G' using $O(\log n)$ memory. This is easy: just answer "yes" if there is such an edge in G, or if $u = t$ or $v = s$, and answer "no" otherwise.

8.8 To show that 2-UNSAT is in NL, we use the reduction to Reachability discussed in Section 4.2.2. A witness that a 2-SAT formula ϕ is unsatisfiable is a path from some variable x to $\overline{x}$, and from $\overline{x}$ back to x.

Given read-only access to ϕ, we can check that each step in this path follows from one of its clauses, and it only takes $O(\log n)$ memory to keep track of the current variable in this path. Thus 2-UNSAT is in NL.

To show that 2-UNSAT is NL-complete, we reduce REACHABILITY to 2-UNSAT. Given a directed graph G, we will generate a 2-SAT formula ϕ with a Boolean variable for each vertex v. Now, for each edge $u \to v$, include the 2-SAT clause $(\overline{u} \vee v)$. This clause can be thought of as a pair of implications: $u \Rightarrow v$, and $\overline{v} \Rightarrow \overline{u}$. Note that setting variables true only forces other variables to be true, and setting variables false only forces other variables to be false.

Now suppose the original REACHABILITY problem asked whether there is a path from s to t in G. With the 2-SAT clauses we have so far, such a path exists if and only if there is a chain of implications from s to t, and a path in the opposite direction from $\overline{t}$ to $\overline{s}$. By adding a few more clauses, we can cause our formula to be unsatisfiable if and only if a path from s to t exists. There are several ways to do this. One is to add an additional variable x, along with clauses

$$(\overline{x} \vee s), (\overline{t} \vee \overline{x}), (x \vee \overline{t}), (s \vee x).$$

Then if there is a path from s to t in G, there is a contradictory loop

$$x \Rightarrow s \Rightarrow t \Rightarrow \overline{x} \Rightarrow \overline{t} \Rightarrow \overline{s} \Rightarrow x.$$

Conversely, setting x true forces s, and the set of all variables reachable from s, to be true—and forces t, and the set of all variables from which we can reach t, to be false. But if t cannot be reached from s then these sets are disjoint, and no contradiction results.

This reduces REACHABILITY to 2-UNSAT, the question of whether a 2-SAT formula is unsatisfiable. Moreover, the reduction consists of local gadgets, and can easily be carried out in L. Specifically, we can answer yes-or-no questions about the formula ϕ, telling whether a given clause exists, given read-only access to the REACHABILITY instance (G, s, t). This proves that 2-UNSAT is NL-complete, and that 2-SAT is coNL-complete. The Immerman–Szelepcsényi Theorem tells us that coNL = NL, so we're done.

8.9 We prove that CYCLIC GRAPH, of whether a cycle exists, is NL-complete, and then use the Immerman–Szelepcsényi Theorem. First, CYCLIC GRAPH is in NL, since a witness consists of a path of length 1 or more which returns to its starting point. We can remember the starting point and the current location with $\log_2 n$ bits each, for a total of $O(\log n)$.

To show that CYCLIC GRAPH is NL-complete, we will reduce REACHABILITY to it. Given a graph $G = (V, E)$ and a pair of vertices s and t, we will generate a graph $G' = (V', E')$ which has a cycle if and only if there is a path from s to t in G. First, let $V' = V \times \{1, \ldots, n\}$, and let E' contain an edge from (u, i) to (v, j) if and only if $j = i + 1$ and there is an edge from u to v in E. Then, for each i, add an edge from (t, i) to $(s, 1)$. Given read-only access to G, it is easy to tell if two vertices of G' are connected by a single step, so this reduction is in L.

8.10 Turn the automaton's state space S into a graph by drawing an edge $s \to s'$ whenever there is some input symbol for which the transition from s to s' is allowed. Formally, if there exists an $a \in \Sigma$ such that $s' \in \delta(s, a)$. Add an additional vertex s_{yes}, and draw an edge from each $s \in S_{\text{yes}}$ to s_{yes}. Then this problem is an instance of REACHABILITY, so it is in NL.

Conversely, this problem is NL-complete because we can reduce REACHABILITY to it. Given an instance (G, s, t), we can easily define an NFA whose states are the vertices of G and whose transitions are its edges. We then set $s_0 = s$ and $S_{\text{yes}} = \{t\}$.

The variant where the automaton is a DFA is still NL-complete. The only difference is that, if a vertex in G has d edges going out from it, we now need d different input symbols to make all these transitions allowed. So, we use an input alphabet Σ whose size is the maximum out-degree.

One way to get an L-complete variant is to limit ourselves to an input alphabet consisting of a single symbol, $\Sigma = \{a\}$. Now this problem is just ONE-OUT REACHABILITY in disguise, and is L-complete under simple gadget reductions. Another way is to ask whether A accepts a specific word w, which is given as part of the problem's input. Then the NFA and DFA versions are essentially REACHABILITY and ONE-OUT REACHABILITY respectively.

8.11 The equation holds because `MiddleFirst` calls itself $2n$ times, twice for each possible midpoint, looking for paths of length $\ell/2$. Iterating the equation until we reach the base case $T(n,1)=1$ then gives

$$T(n,\ell) = 2n\,T(n,\ell/2) = (2n)^2\,T(n,\ell/4) = \cdots = (2n)^{\log_2 \ell}\,T(n,1) = \ell\, n^{\log_2 \ell}\,.$$

Setting $\ell = n$ gives $T(n,n) = n^{\log_2 n+1}$.

8.12 Being in NSPACETIME$(f(n), g(n))$ means that there is a single nondeterministic algorithm that works in $O(f(n))$ space and $O(g(n))$ time. Being in NSPACE$(f(n)) \cap$ NTIME$(g(n))$, on the other hand, means that there is an algorithm that works in $O(f(n))$ space, and another, possibly different, algorithm that works in $O(g(n))$ time. Since optimizing an algorithm for space often makes it inefficient in time and vice versa, there might not be a single algorithm that satisfies both bounds.

Now, a problem in NSPACETIME$(f(n), g(n))$ corresponds to a REACHABILITY problem in a state space of size $2^{O(f(n))}$. But the length of the path corresponds to the computation time, so we only care about paths of length $O(g(n))$. Reviewing the proof of Savitch's Theorem, we see that applying middle-first search gives a stack of depth $O(\log g(n))$, since each depth of recursion divides the length of the path by 2. Each state we store on the stack again takes $O(f(n))$ bits to record, so the total size of the stack is $O(f(n)\log g(n))$.

As shown in Problem 8.11, the running time of `MiddleFirst` is the size of the state space, $2^{O(f(n))}$, raised to the depth of the stack $O(\log g)$. This gives a total running time of $2^{O(f(n)\log g(n))} = g(n)^{O(f(n))}$.

8.16 One way to solve this is to first convert all the gates to NAND gates, using de Morgan's laws and adding NOT gates if necessary. Then each NAND gate is simply a vertex with edges pointing to its inputs: a NAND gate is true if at least one of its inputs is false, and a vertex is winning if and only if at least one of the vertices we can move to is losing. Each false input becomes a dead-end vertex, while each true input becomes a vertex with one outgoing edge pointing to a dead-end vertex.

Alternately, suppose there are no NOT gates, so that the circuit consists just of ANDs and ORs. Let's again call the first and second players the Prover and the Skeptic, since their goals are to show that the output is `true` or `false` respectively. At each OR gate it should be the Prover's turn, since she can show that the OR is true by moving to a true input. Similarly, at each AND gate it should be the Skeptic's turn, since she can prove that the AND is false by moving to a false input. If the AND and OR gates don't alternate, we can switch whose turn it is by adding "dummy" vertices with one edge in and one edge out. These dummy vertices are just NOT gates in disguise, which switch the roles of Prover and Skeptic; in fact, they are the same NOT gates we would add to turn everything into NAND gates in the previous approach.

8.22 Under optimal play, each vertex is a Win, a Loss, or a Draw, where a Draw means that the game will go on forever without either player getting stuck. We can label the vertices using the following rules.

1. If every outgoing neighbor of v is a Win—including if v has no outgoing neighbors—then label v a Loss.
2. If v has at least one outgoing neighbor which is a Loss, label v a Win.

Iterate by sweeping through the vertices and applying these rules wherever they can be applied. (In a sense, this is a dynamic programming algorithm, since whenever we learn that a vertex is a Win or a Loss we remember that in future calculations.) As soon as a sweep results in no new changes, we have reached a fixed point. Then, label all the remaining unlabeled vertices as Draws.

For the corollary, if the graph is directed and acyclic, there's no chance that we will visit any vertex twice, so this restriction might as well not be there.

8.23 Consider a board which is filled entirely with black and white stones. One way to prove that there are no draws is to start at the left corner and follow a path where we keep black and white stones to our left and right respectively, as shown in Figure 8.28. Treat the regions to the upper left and lower left as black and white respectively. The path cannot branch or loop, so it must emerge either on the upper right or lower right side. If the upper right, the White stones to its right form a path from lower left to upper right, and White has won. If the lower right, the Black stones to its left form a path form upper left to lower right, and Black has won.

Now suppose that the second player has a winning strategy. The first player can steal it by making a throwaway move, placing her first stone anywhere, and then pretending that it isn't there. She then responds to the second player according to the winning strategy, as if the second player went first. Each time the winning strategy tells her to place a stone where she already has one, she makes another throwaway move. This continues until she wins or the board is full, in which case the throwaway stone counts as her last move. By assumption, at some point she has a winning path even without her throwaway stone. The only other possibility is that the throwaway stone helps her form a winning path earlier. In either case she wins, contradicting our assumption.

The key to this argument is that having an extra stone of your color on the board can only help you, and can only hurt your opponent. In contrast, in Chess there are positions where being forced to move is to your disadvantage. In Go, players have the option of passing, so it is never to your disadvantage to be the current player, but there can also be draws.

8.24 A game where we are not allowed to return to any previous position, as in the "superko rule," is essentially a game of GEOGRAPHY, except that the graph is exponentially large, with one vertex for every possible game position. (We can change whatever criterion the game uses to define a win or a loss to the one in GEOGRAPHY, where whichever player gets stuck loses, by allowing no moves from a losing endgame while adding one more move from a winning one.) Since GEOGRAPHY on polynomial-size graphs is PSPACE-complete, such a game is potentially EXPSPACE-complete.

Without this restriction, we can follow the same strategy as in Problem 8.22, labeling each game position as a Win, Loss, or Draw. The time it takes to do this is polynomial in the number of game positions, so we can solve such a game in EXPTIME. We can implement the simple ko rule by keeping track of both the current position and the previous one; this just squares the size of the graph, so we again get EXPTIME with a larger exponent.

In general, any ko rule which only requires us to remember only a constant, or even polynomial, number of game positions—such as a rule that we cannot visit any position we have seen in the last k moves

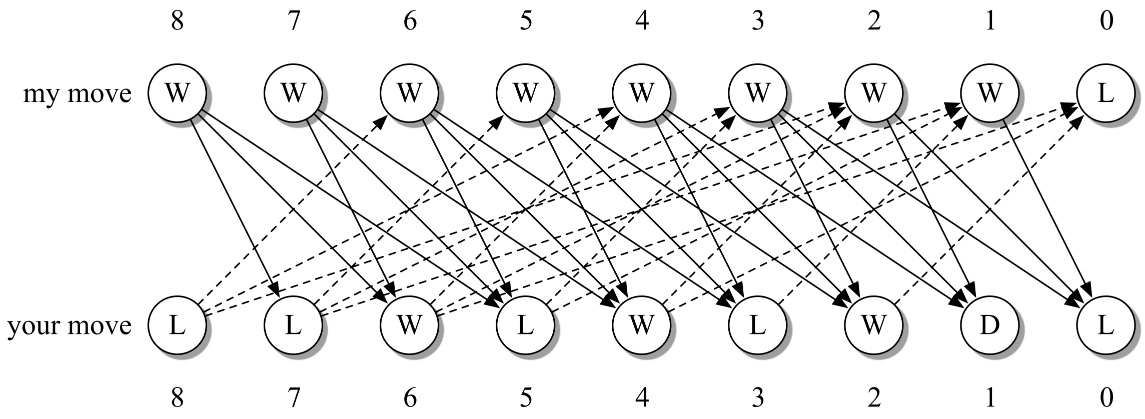


FIGURE 8.1: The graph corresponding to the BEANPILE example. There is one vertex for each combination of whose move it is and how many beans remain in the pile, and the vertices are labeled Win, Loss, or Draw. For instance, if there are 8 beans and it is my move, I can win by removing 1 or 3 beans.

for some $k = \text{poly}(n)$—can be handled in EXPTIME. The problem with the "superko rule" is that, like the restricted form of GEOGRAPHY, it requires us to remember the entire past history of the game.

8.26 Let's encode S and T as strings of n bits, where $s_i = 1$ if I can remove i beans and $s_i = 0$ if I can't, and similarly for T. Then the input to BEANPILE has $\Theta(n)$ bits. Each position consists of the number of beans x remaining in the pile and a bit describing whose turn it is, so there are $2(n+1)$ different positions. This gives us an acyclic graph with $2(n+1)$ vertices as shown in Figure 8.1.

Using dynamic programming, we can label each position as a win, loss, or draw: first, the positions where the pile is empty are losses for the current player, since the other player just won. Then a vertex is a win if it has an edge pointing to at least one loss, a loss if every edge points to a win, and a draw if it has no outgoing edges. (This is just like the algorithm for GEOGRAPHY on an acyclic graph in Problem 8.22, except that if the pile is nonempty and the current player has no legal move, we call it a draw rather than a loss.) Since there are $\text{poly}(n)$ positions, BEANPILE is in P.

8.28 The state space of a sliding block puzzle has N states, where $\log N = \text{poly}(n)$. Let s and t denote the initial and final positions respectively. As discussed in the text, we can tell in PSPACE whether there is a path from s to t. Specifically, we can solve the problem $R(s,t,\ell)$ of whether there is a path from s to t of length ℓ or less. Using $\log_2 N = \text{poly}(n)$ steps of binary search, we can find the length ℓ of the shortest such path.

Start by printing s. Then set $u = s$, and do the following loop. Go through all possible positions v we could move to from u, until we find one such that $R(u,t,\ell-1)$ is true. This v lies on one of the shortest paths from u to t; so print v, set $u = v$, and decrement ℓ. Continue this loop until $\ell = 0$ and $u = t$.

Chapter 9

Optimization and Approximation

9.6 Let $W(j,v)$ denote the weight of the set $S_{j,v}$, or $W(j,v)=\infty$ if no such set exists. Note that $W(j,v)$ is an $\ell \times \ell\, v_{\max}$ table of numbers, each of which corresponds to a subproblem we have to solve. The value $V^\star$ of the optimal knapsack is found in the top row of this table, $V^\star = \max\{v : W(1,v) \le W\}$. The bottom row of the table is given by

$$W(\ell,v)=\begin{cases} w_\ell & \text{if } v = v_\ell \\ \infty & \text{otherwise}. \end{cases}$$

The complete table can then be calculated by dynamic programming,

$$W(j,v)=\min\Big(W(j+1,v),\, w_j + W(j+1, v-v_j)\Big)$$

where $W(j,v)=\infty$ if $v<0$. We can fill in the table in $O(\ell^2 v_{\max})$ time, and then scan the top row to find $V^\star$.

However, this is not a polynomial-time algorithm for KNAPSACK since the size n of the instance is given by the number of bits. If each v_i has b bits then $n=\ell b$. Then $v_{\max}$, and therefore our running time, is exponential as a function of b. As for SUBSET SUM and INTEGER PARTITIONING, KNAPSACK is in P if the v_i are poly(n)—that is, if $b = O(\log n)$, or if the v_i are given in unary instead of binary.

9.7 Let $v_{\max} = \max\{v_1,\ldots,v_n\}$ as before, and let $\varepsilon > 0$. As we will see, the scaling factor we want is $K = \ell/(\varepsilon v_{\max})$. We create a new instance with truncated values

$$v_i' = \left\lfloor \frac{\ell v_i}{\varepsilon v_{\max}} \right\rfloor . \tag{a}$$

Using the dynamic programming algorithm of Problem 9.6, we can solve this truncated instance in time $O(\ell^2 v'_{\max}) = O(\ell^3/\varepsilon)$, i.e., in time polynomial in n and $1/\varepsilon$. If S' denotes the set returned by the dynamic programming algorithm, we need to show that S' is a $(1-\varepsilon)$-approximation for the original problem.

Given a set S, let $V(S)=\sum_{i\in S} v_i$ and $V'(S)=\sum_{i\in S} v_i'$ denote its value with respect to the original and the truncated values respectively. Since (a) implies

$$v_i - \frac{\varepsilon v_{\max}}{\ell} \le (\varepsilon v_{\max}/\ell) v_i' \le v_i ,$$

we have, for any set S,

$$V(S) - \varepsilon v_{\max} \le \frac{\varepsilon v_{\max}}{\ell} V'(S) \le V(S). \tag{b}$$

Now let $S^\star$ denote the optimal solution to the original problem. Since S' is optimal for the truncated problem, we have

$$V'(S') \ge V'(S^\star).$$

But combining this with (b) gives

$$V(S') \ge \frac{\varepsilon v_{\max}}{\ell} V'(S') \ge \frac{\varepsilon v_{\max}}{\ell} V'(S^\star) \ge V(S^\star) - \varepsilon v_{\max} \ge (1-\varepsilon) V(S^\star),$$

where in the last inequality we assumed without loss of generality that $V(S^\star) \ge v_{\max}$. This follows from the assumption that $v_{\max} \le W$, i.e., that none of our objects is too heavy to carry. Thus S' is a $(1-\varepsilon)$-approximation.

9.8 Let u_t denote the number of uncovered elements just after the greedy algorithm has taken its tth step. Since the optimal cover has size k, there must be a some set S_j in the optimal cover, not yet taken by the greedy algorithm, which covers at least u_t/k of these elements. Therefore, the next step of the greedy algorithm reduces the number of uncovered elements to

$$u_{t+1} \le u_t - u_t/k = (1 - 1/k) u_t .$$

We have $u_0 = m$, and so

$$u_t \le m(1-1/k)^t < m \mathrm{e}^{-t/k} .$$

The algorithm stops as soon as $u_t = 0$, at which point it has produced a cover of size t. Since u_t is integer-valued, we can also say that it stops as soon as $u_t < 1$. We have shown that this happens when $t = k \ln m$, if not before.

9.9 Consider a random assignment $x = (x_1, \ldots, x_n)$ where each variable x_i independently takes on the value `true` or `false` with probability 1/2. Let $X_i = 1$ if x satisfies clause i and $X_i = 0$ if it doesn't. Then $\mathbb{E}(X_i) = 1 - 2^{-k}$ since only one out of 2^k assignments of the literals in each clause violates the clause. The number of satisfied clauses is $X = \sum_i X_i$, and linearity of expectation gives

$$\mathbb{E}(X) = \sum_i \mathbb{E}(X_i) = m(1 - 2^{-k}). \tag{a}$$

Now either all assignments satisfy precisely $\mathbb{E}(X)$ clauses, or some satisfy less and others more than that. In any case there must be at least one assignment that satifies $\mathbb{E}(X)$ or more clauses.

An algorithm that finds this assignment in polynomial time will somehow set variables one at a time without backtracking. Let's assume that we are about to set variable x_i, and let C_p denote the set of all clauses that contain x_i and C_n the set of clauses that contain $\overline{x}_i$. If we set x_i to `true`, we can remove all clauses C_p from the formula and we can remove the literal $\overline{x}_i$ from all clauses C_n. If we set x_i to `false`, the same reduction applies with C_p and C_n exchanged. In the course of fixing variables we are therefore dealing with formulae made of clauses of sizes between 0 and k. If $C(\phi)$ is the current set of clauses, the expected number u of clauses that is unsatisfied by a random assignment of the remaining variables reads

$$u(\phi) = \sum_{c \in C(\phi)} 2^{-|c|},$$

where $|c|$ denotes the size of clause c, i.e., the number of literals in c. Setting a variable x_i `true` or `false` results in two different formulae $\phi[x_i = \texttt{true}]$ and $\phi[x_i = \texttt{false}]$, and the key idea is to set x_i such that u is minimized.

Let's call this algorithm `Greedy-SAT`, since it minimizes the expected number of unsatisfied clauses at each step. `Greedy-SAT` is a $(1-2^{-k})$-approximation to MAX-k-SAT.

To prove this we will show that in each iteration the expectation $\mathbb{E}(X)$, conditioned on the variables that have already been assigned, does not decrease. The approximation ratio the follows from the initial value (a) of $\mathbb{E}(X)$.

Suppose we set $x_i = \texttt{true}$. This means that

$$u\left(\phi[x_i = \texttt{false}]\right) - u\left(\phi[x_i = \texttt{true}]\right) = \sum_{c \in C_p} 2^{-|c|+1} - \sum_{c \in C_n} 2^{-|c|+1} \geq 0\,.$$

The contribution to $\mathbb{E}(X)$ from a clause $c \in C_p$ changes from $1 - 2^{-|c|}$ to 1, the contribution from a clause $c \in C_n$ changes from $1 - 2^{-|c|}$ to $1 - 2^{-|c|+1}$. The net change on $\mathbb{E}(X)$ is

$$\sum_{c \in C_p} 2^{-|c|} - \sum_{c \in C_n} 2^{-|c|}$$

which is always nonnegative according to our decision $x_i = \texttt{true}$. The case $x_i = \texttt{false}$ works exactly the same with C_p and C_n exchanged.

9.12 The claim is trivially true for $d = 1$. So let us assume that (a) and (b) are true for $d - 1$, and try to prove them for d. In the initial phase, the constraint $\varepsilon x_{d-1} = x_d$ is tight, and according to our pivot rule we will not flip to $x_d = 1 - \varepsilon x_{d-1}$ until it is impossible to flip any other constraint of lower index. But at the same time, x_d increases with each step, so the initial phase consists of maximizing εx_{d-1} over the projection of the Klee–Minty cube onto the first $d-1$ coordinates, $(x_1, \ldots, x_{d-1})$. This projection is exactly the $(d-1)$-dimensional Klee–Minty cube. Thus by induction, this part of the walk visits every vertex such that $\varepsilon x_{d-1} = x_d$, and stops at $(0, \ldots, 0, 1, \varepsilon)$.

The next vertex is $(0, \ldots, 0, 1, 1-\varepsilon)$, so now the constraint $x_d = 1 - \varepsilon x_{d-1}$ is tight. Now the pivot rule tries to minimize x_{d-1} on the $(d-1)$-dimensional cube. By induction on (b), the second half of the walk visits all vertices with $x_d = 1 - \varepsilon x_{d-1}$.

Combining the first and second halves of the walk visits all 2^d vertices, and we have proved (a). Flipping this argument proves (b) as well.

9.13 It is easy to verify (9.59) for $d = 2$. So let us assume that it is true for $d - 1$. Let $\mathbf{u}, \mathbf{v}$ be vertices of a 0-1 polytope P in d dimensions with $\delta(\mathbf{u}, \mathbf{v}) \geq d$. Because of the symmetry of the hypercube, we can assume that $\mathbf{v} = \mathbf{0}$.

Let us assume that $\mathbf{u}$ has at least one zero component, say $u_i = 0$. Then $\mathbf{0}$ and $\mathbf{u}$ share the facet

$$F_i = P \cap \{\mathbf{x} \in \mathbb{R}^d : x_i = 0\}\,,$$

which is a $(d-1)$-dimensional 0-1 polytope. By induction, $\delta(\mathbf{u},\mathbf{0}) \le d-1$. Hence we can assume that $\mathbf{u}=\mathbf{1}$.

Now let us assume that $\mathbf{1}$ has a neighbor $\mathbf{w}$ which has $k>1$ zero components. By the same reasoning as above we have

$$\delta(\mathbf{0},\mathbf{1}) \le \delta(\mathbf{0},\mathbf{w}) + \delta(\mathbf{w},\mathbf{1}) \le (m-k)+1\,.$$

Hence we can conclude that all neighbors of $\mathbf{1}$ have a single zero component. We also know that each vertex of a d-dimensional polytope has at least d neighbors. Hence $\mathbf{1}$ has exactly d neighbors $\mathbf{w}_i = \mathbf{1}-\mathbf{e}_i$, $i=1,\ldots,m$. Now $\mathbf{0}$ and $\mathbf{w}_i$ share the face F_i, hence $\delta(\mathbf{0},\mathbf{w}_i) \le d-1$ by induction. But since we started with $\delta(\mathbf{0},\mathbf{1}) \ge d$, we know that $\delta(\mathbf{0},\mathbf{w}_i) = d-1$, and hence by induction that F_i is the $(d-1)$-dimensional hypercube

$$F_i = \{\mathbf{x} \in \mathbb{R}^d : 0 \le x_{j\neq i} \le 1,\, x_i = 0\}.$$

Hence $\delta(\mathbf{0},\mathbf{1}) \le d$ on P, with equality if and only if P is the hypercube.

For the proof of (9.60) note that every vertex of a d-dimensional polytope lies in at least d facets. If $m<2d$, any two vertices must at least one common facet. Hence $\Delta(d,m) \le \Delta(d-1,m-1)$. Iterating this equation $2d-m$ times gives (9.60).

9.16 We append m new variables to $\mathbf{x}$, giving

$$\mathbf{x}' = (x_1,\ldots,x_d,x_{d+1},\ldots,x_{d+m})^T\,.$$

We also append the $m\times m$ identity matrix $\mathbb{1}$ to the columns of $\mathbf{A}$, giving $\mathbf{A}' = (\mathbf{A}|\mathbb{1})$. Then the linear program

$$\min_{\mathbf{x}} \left(x_{d+1}+x_{d+2}+\ldots+x_{d+m}\right) \quad \text{subject to} \quad \mathbf{A}'\mathbf{x}' = \mathbf{b} \quad \text{and} \quad \mathbf{x}' \ge 0$$

has the feasible solution $x_1 = \cdots = x_d = 0$ and $x_{d+1} = b_1,\ldots,x_{d+m} = b_m$. The minimum cannot be less than zero since $\mathbf{x}' \ge 0$, and it is zero if and only if the first d components of $\mathbf{x}'$ give a feasible solution $\mathbf{x}$ to the original linear program (9.61).

9.17 A given vertex $\mathbf{v}$ satisfies d linearly independent constraints of the form $\mathbf{a}_j^T\mathbf{x} = b_j$. Let $\mathbf{A}'$ be the $d\times d$ matrix consisting of these rows, and let $\mathbf{b}' \in \mathbb{R}^d$ be the vector consisting of these entries of $\mathbf{b}$. Then $\mathbf{A}'\mathbf{v} = \mathbf{b}$, so

$$v_i = \frac{\det \mathbf{A}''_i}{\det \mathbf{A}'}\,.$$

where $\mathbf{A}''_i$ is the matrix formed by replacing the ith column of $\mathbf{A}'$ with $\mathbf{b}'$.

The determinant of $\mathbf{A}'$ is the volume of the parallelepiped spanned by its row vectors $\mathbf{a}'_j$, which is at most the product of their lengths. So for the denominator, we have

$$D^2 = \left|\det \mathbf{A}'\right|^2 \le \prod_{j=1}^{d} \left|\mathbf{a}'_j\right|^2 \le \prod_{j=1}^{d}\sum_{i=1}^{d} {a'_{ij}}^2 \le 2^{2n}\,.$$

Thus $D = |\det \mathbf{A}'| \le 2^n$. The same reasoning applies to the nominator $\det \mathbf{A}''_i$, so $|u_i| \le 2^n$ for each i.

9.18 The upper bound simply follows from $|x_i| \le 2^n$ for all i, and the fact that $|c_i| \le 2^n$ by the definition of n.

For the second part, we know that each vertex is rational with denominator $D \le 2^n$. The same is true of $\mathbf{a}_j^T \mathbf{v}$, so if it does not equal the integer b_j, it must differ from it by at least $1/D \ge 2^{-n}$.

It follows that any constraint whose "looseness" $b_j - \mathbf{a}_j^T \mathbf{v}$ is less than 2^{-n} is in fact tight at the optimum. Thus if we can determine the optimal vertex to within 2^{-2n}, say, we can check each constraint to see which ones are tight. We can then find the optimal vertex exactly by taking the intersection of the tight constraints.

Similarly, for any two vertices $\mathbf{v}, \mathbf{v}'$, $\mathbf{c}^T\mathbf{v} - \mathbf{c}^T\mathbf{v}'$ is the difference of two rational numbers with denominators $D, D' \le 2^n$, which is rational with denominator at most $DD' \le 2^{-2n}$. If all we can do is approximate the optimal value $\mathbf{c}^T\mathbf{x}$, we try tightening each constraint, reducing b_j by 2^{-2n}, say. (In that case, b_j is no longer an integer, but we can multiply $\mathbf{A}$ and $\mathbf{b}$ by 2^{-2n} to compensate, and this just multiplies n by a constant.) We then approximate $\mathbf{c}^T\mathbf{x}$ within $\varepsilon = 2^{-3n}$. Tightening a constraint forces us to a different vertex of lower quality if and only if that constraint was tight in the first place. This again allows us to identify the tight constraints and compute $\mathbf{x}$.

9.19 If there is no such set of vertices then all the vertices are contained in a $(d-1)$-dimensional subspace. Since P is the convex hull of the vertices, in that case P is contained in that subspace. On the other hand, if there is such a set of vertices, then by convexity P contains their convex hull.

According to Problem 9.17, each $\mathbf{v}_i$ is an integer vector divided by denominator $D_i \le 2^n$. The determinant in (9.65) is then rational with denominator at most $\prod_{j=0}^{d} D_j \le 2^{(d+1)n}$. Therefore, if it not zero its absolute value is at least $2^{-(d+1)n}$. Dividing by $d! \le d^d = 2^{-O(d\log d)}$ gives

$$\text{volume}(S) \ge \frac{2^{-(d+1)n}}{d^d} = 2^{-O(dn)} .$$

9.20 A real symmetric $d \times d$ matrix has real eigenvalues, and its eigenvectors form an orthonormal basis. Let $\lambda_1, \ldots, \lambda_d$ denote the eigenvalues of $\mathbf{Q}$, and let $\mathbf{u}_1, \ldots, \mathbf{u}_d$ denote the corresponding eigenvectors. Since the $\mathbf{u}_k$ form a basis, we can write any vector $\mathbf{x} \in \mathbb{R}^d$ as $\mathbf{x} = \sum_k x_k \mathbf{u}_k$. This allows us to write

$$\mathbf{x}^T\mathbf{Q}\mathbf{x} = \sum_{jk} x_j x_k \, \mathbf{u}_j^T \mathbf{Q} \mathbf{u}_k = \sum_{jk} x_j x_k \, \lambda_k \mathbf{u}_j^T \mathbf{u}_k = \sum_k x_k^2 \lambda_k .$$

If $\lambda_k \ge 0$ for all k, then $\mathbf{x}^T\mathbf{Q}\mathbf{x} \ge 0$. Conversely, if there is a $\lambda_k < 0$, we have $\mathbf{u}_k^T \mathbf{Q} \mathbf{u}_k = \lambda_k < 0$. Thus (1) holds if and only if (2) does.

If $\mathbf{Q} = \mathbf{M}^T\mathbf{M}$, then

$$\mathbf{x}^T\mathbf{Q}\mathbf{x} = \mathbf{x}^T\mathbf{M}^T\mathbf{M}\mathbf{x} = |\mathbf{M}\mathbf{x}|^2 \ge 0 .$$

Thus (3) implies (1).

Finally, we show that (2) implies (3). Let $\mathbf{U}$ be the matrix whose columns are the eigenvectors $\mathbf{u}_k$. Since they are orthonormal, we have $\mathbf{U}^T\mathbf{U} = \mathbb{1}$ and

$$\mathbf{U}^T\mathbf{Q}\mathbf{U} = \text{diag}(\lambda_1, \ldots, \lambda_d) ,$$

where $\text{diag}(\lambda_1, \ldots, \lambda_d)$ denotes the diagonal matrix with entries $\lambda_1, \ldots, \lambda_d$. Since $\lambda_k \ge 0$, we can take the square root,

$$\mathbf{U}^T\mathbf{Q}\mathbf{U} = \mathbf{D}\mathbf{D} \quad \text{where} \quad \mathbf{D} = \text{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d}\right) .$$

Multiplying this equation on the left and right by $\mathbf{U}$ and $\mathbf{U}^T$ respectively gives

$$\mathbf{Q} = \mathbf{UDDU}^T = (\mathbf{DU}^T)^T(\mathbf{DU}^T).$$

Thus (3) holds with $\mathbf{M} = \mathbf{DU}^T$.

9.21 We clearly have $S_r^d = S_1^d r^d$, so we can focus on calculating S_1^d. Direct evaluation of I is easy because the integral factorizes:

$$I = \left(\int_{-\infty}^{\infty} \mathrm{e}^{-x_1^2}\,\mathrm{d}x_1\right)\left(\int_{-\infty}^{\infty} \mathrm{e}^{-x_2^2}\,\mathrm{d}x_2\right)\cdots\left(\int_{-\infty}^{\infty} \mathrm{e}^{-x_d^2}\,\mathrm{d}x_d\right) = \pi^{n/2}.$$

On the other hand, in radial coordinates,

$$\begin{aligned} I &= \int_0^\infty \frac{\mathrm{d}\,\mathrm{volume}(S_r^d)}{\mathrm{d}r}\,\mathrm{e}^{-r^2}\,\mathrm{d}r \\ &= \mathrm{volume}(S_1^d)\int_0^\infty d\,r^{d-1}\,\mathrm{e}^{-r^2}\,\mathrm{d}r \\ &= \mathrm{volume}(S_1^d)\,\frac{d}{2}\int_0^\infty t^{\frac{d}{2}-1}\,\mathrm{e}^{-t}\,dt \\ &= \mathrm{volume}(S_1^d)\,\frac{d}{2}\,\Gamma(\frac{d}{2}) \\ &= \mathrm{volume}(S_1^d)\,\Gamma\left(\frac{d}{2}+1\right). \end{aligned}$$

In the third line, we changed the variable of integration from r to $t = r^2$, and in the last line we used the identity $x\Gamma(x) = \Gamma(x+1)$. We then obtain volume(S_1^d) by equating both expressions for I.

Applying the affine transformation $\mathbf{x} \to \mathbf{M}^{-1}\mathbf{x} + \mathbf{r}$ multiplies the volume by $\det \mathbf{M}^{-1} = 1/\det \mathbf{M}$. Since $\mathbf{Q} = \mathbf{M}^T\mathbf{M}$, we have $\det \mathbf{M} = \sqrt{\det \mathbf{Q}}$, and we're done.

9.22 The height of the ellipsoid along the x_1-axis is $1 - r$. To contain the hemisphere, the width of the ellipsoid in the other $d-1$ axes must be $\sqrt{(1-r)^2/(1-2r)}$. The product of this height and these widths gives the ratio of the volume of E' compared to that of the unit sphere. Minimizing the volume with respect to r then gives $r = 1/(d+1)$, a calculus problem which we omit.

Rewriting the ratio between the volumes and using $1 + x \le \mathrm{e}^x$ yields the upper bound:

$$\begin{aligned} \left(\frac{d}{d+1}\right)\left(\frac{d^2}{d^2-1}\right)^{\frac{d-1}{2}} &= \left(1 - \frac{1}{d+1}\right)\left(1 + \frac{1}{d^2-1}\right)^{\frac{d-1}{2}} \\ &\le \mathrm{e}^{-1/(d+1)}\left(\mathrm{e}^{1/(d^2-1)}\right)^{\frac{d-1}{2}} \\ &= \mathrm{e}^{-1/(2d+2)}. \end{aligned}$$

9.25 Note that $p(\mathbf{y}) \geq 0$ for all feasible $\mathbf{y}$, and that $p(\mathbf{y}) = 0$ if and only if the y_i saturate the constraints (9.68), i.e., if $|y_i - 1/2| = y_0$ for all $1 \leq i \leq n$. For any fixed $y_0 > 0$, this means that $\mathbf{y}$ is at one of the corners of a hypercube, where $y_i = 1/2 \pm y_0 z_i$ for some $z_i = \pm 1$. We can interpret $\mathbf{z}$ as a truth assignment $\mathbf{x}$, where $z_i = +1$ if $x_i = \texttt{true}$ and $z_i = -1$ if $x_i = \texttt{false}$. If $\mathbf{x}$ is a satisfying assignment, each clause constraint (9.67) is satisfied. Hence, if ϕ is satisfiable, there is a feasible solution $\mathbf{y}$ with $p(\mathbf{y}) = 0$ and $q(\mathbf{y}) = -y_0^2/2$. In that case, $\min f \leq -y_0^2/2$ for each $y_0 > 0$. Since $f(\mathbf{y}^\star) = 0$, this shows that $\mathbf{y}^\star$ is not a local minimum.

Now suppose that ϕ is not satisfiable. Then there is at least one clause where all the literals are false. Assume without loss of generality that none of the variables in that clause are negated, so that $y_1, y_2, y_3 \leq 1/2$. On the other hand, at least one of them, say, y_1 is greater than or equal to $1/2 - y_0/3$:

$$\frac{1}{2} - \frac{y_0}{3} \leq y_1 \leq \frac{1}{2}.$$

But then y_1 contributes $(y_0 - (y_1 - 1/2))^2 = (8/9)y_0^2$ to p. All the terms in p are nonnegative, so $p(\mathbf{y}) \geq (8/9)y_0^2$. Since we still have $q(\mathbf{y}) \geq -y_0^2/2$, we have $\min f \geq (7/18)y_0^2$ for all $y_0 > 0$, and $\mathbf{y}^\star$ is a local minimum.

9.28 Assume that $\mathbf{x}$ is a vertex that is not half-integral. This implies that

$$V_+ = \left\{ v : \frac{1}{2} < x_v < 1 \right\} \quad \text{and} \quad V_- = \left\{ v : 0 < x_v < \frac{1}{2} \right\}$$

are not both empty. We define two vectors $\mathbf{y}$ and $\mathbf{z}$ as

$$y_v = \begin{cases} x_v + \varepsilon & \text{if } v \in V_+ \\ x_v - \varepsilon & \text{if } v \in V_- \\ x_v & \text{otherwise} \end{cases} \qquad z_v = \begin{cases} x_v - \varepsilon & \text{if } v \in V_+ \\ x_v + \varepsilon & \text{if } v \in V_- \\ x_v & \text{otherwise} \end{cases}$$

Note that $\mathbf{x} \neq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{z}$ by assumption. If we choose $\varepsilon > 0$ small enough, we can guarantee that $0 \leq y_v, z_v \leq 1$ for all v and, in addition,

$$x_u + x_v > 1 \quad \Rightarrow \quad y_u + y_v > 1\,,\ z_u + z_v > 1\,.$$

The case $x_u + x_v = 1$ implies that either $x_u = x_v = \frac{1}{2}$, or $x_u = 0$ and $x_v = 1$, or $u \in V_-$ and $v \in V_+$. But in all three cases we have $x_u + x_v = y_u + y_v = z_u + z_v = 1$. We conclude that for ε small enough, $\mathbf{y}$ and $\mathbf{z}$ are feasible solutions. Hence $\mathbf{x} = (\mathbf{y} + \mathbf{z})/2$ is the convex combination of two feasible vectors and therefore can't be a vertex of the polytope.

9.29 The claim that $S_{\min}$ contains none of V_0 can be written as

$$S_{\min} \subseteq V_1 \cup V_{1/2}\,, \tag{a}$$

and the claim that $S_{\min}$ contains all of V_1 means

$$V_1 \subseteq S_{\min}\,. \tag{b}$$

Let us assume that (a) is true, and that there exists $v \in V_1$ such that $v \notin S_{\min}$. Then all neighbours of v must be in $S_{\min}$, and hence in $V_1 \cup V_{1/2}$. But then we can reduce x_v from $x_v = 1$ to $x_v = 1/2$, thereby reducing the total weight of $\mathbf{x}$. This contradicts the fact that $\mathbf{x}$ is a minimum of the relaxed problem. Hence (a) implies (b).

To prove (a), let's assume that $V_0 \cap S_{\min}$ is not empty. All neighbours of vertices $V_0 \cap S_{\min}$ are in V_1. Edges that run between $V_0 \cap S_{\min}$ and $V_1 \cap S_{\min}$ have both endpoints covered by $S_{\min}$, while those edges that run between $S_{\min} \cup V_0$ and $V_1 - S_{\min}$ have only one endpoint covered. If $|V_0 \cap S_{\min}| \geq |V_1 - S_{\min}|$, we can create a new cover S' with $|S'| \leq |S_{\min}|$ by uncovering all vertices in $V_0 \cap S_{\min}$ and covering those in $V_1 - S_{\min}$. But S' satisfies (a).

Hence we can assume $|V_0 \cap S_{\min}| < |V_1 - S_{\min}|$. But since $S_{\min}$ is a cover, there can't be any edges between $V_1 - S_{\min}$ and $V_0 - S_{\min}$, so all edges from $V_1 - S_{\min}$ to V_0 end in $V_0 \cap S_{\min}$. Moreover, every vertex $u \in V_1 - S_{\min}$ has to have at least one edge connecting it to some $v \in V_0 \cap S_{\min}$, since otherwise we could reduce x_u to $1/2$ and thus reduce the total weight of $\mathbf{x}$.

But if $|V_0 \cap S_{\min}| < |V_1 - S_{\min}|$, at least one vertex $v \in V_0 \cap S_{\min}$ must be connected to two or more vertices $v_1, v_2 \in V_1 - S_{\min}$. We then can reduce $x_{v_1}, x_{v_2}, \ldots$ from 1 to $1/2$ and increase x_v from 0 to $1/2$, thus reducing the total weight of $\mathbf{x}$. Again this contradicts our assumption that $\mathbf{x}$ is a minimal solution to the relaxed problem, and this proves (a).

9.31 If the network has m edges, we think of the flow as an m-dimensional vector $\mathbf{f}$. For each vertex v other than s and t, the flow into v must equal the flow out of v; that is,

$$\sum_{e \in \text{in}(v)} f_e - \sum_{e \in \text{out}(v)} f_e = 0\,.$$

But this can be written as

$$\mathbf{A}'\mathbf{x} = \mathbf{b}\,,$$

where $\mathbf{A} = (a_{ev})$ is the incidence matrix of the graph,

$$a_{ev} = \begin{cases} +1 & \text{if } e \text{ is an outgoing edge from } v\,, \\ -1 & \text{if } e \text{ is an incoming edge to } v\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

and $\mathbf{A}'$ is its minor where we remove the columns corresponding to s and t. We showed in Problem 9.30 that $\mathbf{A}$ is totally unimodular, and therefore so is $\mathbf{A}'$.

If $\mathbf{c}$ is the vector of edge capacities then the other constraint is $\mathbf{f} \leq \mathbf{c}$, i.e., $f_e \leq c_e$ for all e. Finally, we want to maximize the value of the flow. We can write this as

$$\sum_{e \in \text{out}(s)} f_e = \mathbf{d}^T \mathbf{f},$$

where $\mathbf{d}$ is the column of $\mathbf{A}$ corresponding to s. Since $\mathbf{A}$ is totally unimodular, the resulting linear program has an integral optimum as long as the edge capacities c_e are integers.

9.32 A path from s to t is a subset of edges such that s has exactly one outgoing edge, t has exactly one incoming edge, and each other vertex has as many outgoing edges as it has incoming edges. But if we represent this subset as its characteristic vector, it is simply a flow **f** as in the previous problem, where $0 \le f_e \le 1$ for each edge e, and where the total value of the flow is 1. The constraint on **f** is then

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,,$$

where $\mathbf{A} = (a_{ev})$ is the incidence matrix of the graph, and where the components of **b** are zero except $b_s = 1$ and $b_t = -1$. The total length of the path is $\mathbf{c}^T\mathbf{f}$. Hence SHORTEST PATH can be written as

$$\min_{\mathbf{f}} \mathbf{c}^T\mathbf{f} \quad \text{subject to} \quad \mathbf{A}\mathbf{f} = \mathbf{b}, \mathbf{f} \ge \mathbf{0}\}\,.$$

As before, we showed in Problem 9.30 that **A** is unimodular. So this LP relaxation has integral solutions, corresponding to genuine paths.

The dual of this problem reads

$$\max_{\mathbf{y}} \mathbf{b}^T\mathbf{y} \quad \text{subject to} \quad \mathbf{A}^T\mathbf{y} \le \mathbf{c}, \mathbf{y} \ge \mathbf{0}\}\,,$$

which we can also write as

$$\max_{\mathbf{y}} (y_s - y_t) \quad \text{subject to} \quad \left| y_u - y_v \right| \le c_{uv} \quad \text{for all} \quad (u,v) \in E\,.$$

This has a nice interpretation in terms of a bead-string network: we want to pull beads s and t as far apart as possible, but the distance between any two beads can't be larger than the length of the string that connects them. When we do this, the edges on the shortest path—or, if this is not unique, on all the shortest paths—are pulled taut.

9.33 The coefficient matrix of (9.69), not counting $A_{ij} \ge 0$, is of the form

$$\left(\begin{array}{cccc|cccc|c|cccc}
1 & 1 & \cdots & 1 & & & & & & & & & \\
 & & & & 1 & 1 & \cdots & 1 & & & & & \\
 & & & & & & & & \cdots & & & & \\
 & & & & & & & & & 1 & 1 & \cdots & 1 \\
\hline
1 & & & & 1 & & & & & 1 & & & \\
 & 1 & & & & 1 & & & & & 1 & & \\
 & & \ddots & & & & \ddots & & \cdots & & & \ddots & \\
 & & & 1 & & & & 1 & & & & & 1
\end{array}\right).$$

This is the incidence matrix of the complete bipartite graph $K_{n,n}$, with n vertices on either side and all n^2 possible edges. The rows and columns of A correspond to the vertices on the left and right of $K_{n,n}$ respectively, i.e., the first n and last n rows of this incidence matrix. Each entry a_{ij} corresponds to an edge of $K_{n,n}$, i.e., a column of this incidence matrix, since it participates in one row and one column of A.

Theorem 9.2 then tells us that the polytope of doubly stochastic matrices is integral, so we can reduce ASSIGNMENT to LP. The same is true for maximizing or minimizing any other linear function in this polytope, including MAX-WEIGHT PERFECT MATCHING.

9.34 Suppose that whenever A is doubly stochastic, there is a permutation matrix $M^{(\pi)}$ in its support. Once we show this, we can subtract $aM^{(\pi)}$ from A where $a = \min_i a_{i,\pi(i)}$, and multiply A by a constant to make it doubly stochastic again. This decreases the support of A, so after at most $n^2 - n$ steps A has only n nonzero entries. At this point, A is just a permutation matrix. This shows that A is a convex combination of permutation matrices.

To show that there is such an $M^{(\pi)}$, we treat this as an instance of PERFECT BIPARTITE MATCHING. Define a bipartite graph B with n vertices on either side, with an edge from vertex i on the left to vertex j on the right if $A_{ij} > 0$. We will show that B has a perfect matching, connecting each i on the left to $j = \pi(i)$ on the right for some permutation π.

Hall's Theorem (see Problem 3.47) states that B has a perfect matching if and only if every subset S on the left is connected to a set T on the right where $|T| \geq |S|$. Equivalently, let $\mathbf{v}$ be the characteristic vector of S, and let $\operatorname{supp} \mathbf{v} = \{i : w_i > 0\}$ denote its support. Then $S = \operatorname{supp} \mathbf{v}$ and $T = \operatorname{supp} A\mathbf{v}$, so we need to show that $\left|\operatorname{supp} A\mathbf{v}\right| \geq |S|$.

To prove this, note that $A\mathbf{v}$ is the sum, over all $i \in S$, of the ith column of A. Since A is stochastic, each of these colums sums to 1, so $\sum_i (A\mathbf{v})_i = |S|$. However, since the rows of A also sum to 1, for all i we have $(A\mathbf{v})_i \leq 1$. It follows that $\left|\operatorname{supp} A\mathbf{v}\right| \geq |S|$. This completes the proof.

9.35 If P had integral vertices, we could solve MAX 3-MATCHING in polynomial time. We define $w_{ijk} = 1$ if the triple (i, j, k) is compatible and $w_{ijk} = 0$ otherwise, and maximize the total weight $\sum_{ijk} w_{ijk} A_{ijk}$ over P. Since MAX 3-MATCHING is NP-complete, this would imply that P = NP.

For $n = 2$, there are four integral solutions, each one with two 1s at opposite corners of the cube. The following A is triply stochastic, but is not in the convex hull of these points: $A_{1,1,1} = A_{1,2,2} = A_{2,1,2} = A_{2,2,1} = 1/2$, and $A_{2,1,1} = A_{1,2,1} = A_{1,1,2} = A_{2,2,2} = 0$. Indeed, this A is a vertex of P.

9.36 The reduction 3-CYCLE COVER ≤ MIN-3-CYCLE COVER is almost trivial: just map G to a distance matrix via (9.3). For the second step we prove

$$\text{PERFECT 3-MATCHING} \leq \text{3-CYCLE COVER}\,. \qquad \text{(a)}$$

Recall that an instance of PERFECT 3-MATCHING (Section 5.5.2) is given by three sets U, V, W of equal size n and a set $S \subseteq U \times V \times W$ of compatible triplets. PERFECT 3-MATCHING asks whether there is a perfect 3-matching in S, i.e., a subset $T \subseteq S$ that includes each element in U, V, and W exactly once. We will construct a graph $G(V, E)$ from S such that S has a perfect 3-matching if and only if G has a 3-cycle cover. We start with base vertex set $V = U \cup V \cup W$, and for each $(u, v, w) \in S$ we add a gadget made of 9 extra vertices and 18 edges as shown in Figure 9.49. The resulting graph has $|V| = 3n + 9|S|$ vertices and $|E| = 18|S|$ edges and can be constructed in polynomial time.

Each of the gadgets can be covered by 3-cycles in only two different ways: either including all of its base vertices (u, v, w) (grey triangles) or excluding all of them (white triangles). Let us assume that $T \subseteq S$ is a perfect matching. Then each block based on $(u, v, w) \in T$ is covered with grey triangles, all other blocks are covered with white triangles. The result is a complete 3-cycle cover of G.

For the reverse direction, assume that G has a 3-cycle cover. This means each element $u \in U$ is covered by exactly one grey triangle, and the other grey triangles of the corresponding gadget cover elements $v \in V$ and $w \in W$. Each triplet (u, v, w) covered by grey triangles is an element of T, and the triplets not in T belong the gadgets covered by white triangles.

9.37 Since $\delta_e(T)=0$ if $e \in T$, we can assume that $e \notin T$. Then applying (9.70) to $S \cup \{e\}$ and T gives

$$\begin{aligned} f(S\cup\{e\})+f(T) &\geq f\big((S\cup\{e\})\cap T\big)+f\big((S\cup\{e\})\cup T\big) \\ &= f(S)+f(T\cup\{e\}), \end{aligned}$$

which is equivalent to (9.72). For the converse direction, let $T-S=\{e_1,\ldots,e_r\}$. Using (9.72) r times gives

$$\begin{aligned} f(S\cap T)-f(S) &\leq f\big((S\cap T)\cup\{e_1\}\big)-f(S\cup\{e_1\}) \\ &\;\;\vdots \\ &\leq f\big((S\cap T)\cup\{e_1,\ldots,e_r\}\big)-f(S\cup\{e_1,\ldots,e_r\}) \\ &= f(T)-f(S\cup T), \end{aligned}$$

which is equivalent to (9.70).

9.38 Adding edges can only decrease the number of connected components, so r is nondecreasing. If an additional edge e connects two previously unconnected components of $G(V,F)$, then $r(F\cup\{e\})=r(F)+1$. Otherwise, $r(F\cup\{e\})=r(F)$. Hence both sides of

$$r(F\cup\{e\})-r(F)\geq r(H\cup\{e\})-r(H)$$

are 0 or 1, and the only way to violate this inequality would be an edge e that runs between two vertices that are already connected in $G(V,F)$, but not connected in $G(V,H)$. But this can never happen if $F\subseteq H$. Thus $\delta_e(F)=r(F\cup\{e\})-r(F)$ is nonincreasing, and r is submodular.

9.39 $\mathbf{x}\geq 0$ because f is nondecreasing. And for all $T\subset E$ we have

$$\begin{aligned} \sum_{j\in T} x_j &= \sum_{j\in T, j\leq k}\big(f(S^j)-f(S^{j-1})\big) \\ &\leq \sum_{j\in T, j\leq k}\big(f(S^j\cap T)-f(S^{j-1}\cap T)\big) \\ &= f(S^k\cap T)-f(\emptyset) \\ &\leq f(T)-f(\emptyset) \\ &= f(T). \end{aligned}$$

Hence $\mathbf{x}$ is a feasible solution of the primal problem. The dual problem reads

$$\begin{aligned} \text{minimize} \quad & \sum_{S\subset E} f(S)y_S \\ \text{subject to} \quad & \sum_{S:j\in S} y_S \geq c_j\,, \qquad \forall j\in E \\ & \mathbf{y}\geq \mathbf{0}. \end{aligned}$$

The solution $\mathbf{y}$ is a feasible solution of the dual because $\mathbf{y}\geq\mathbf{0}$ and

$$\sum_{S:j\in S} y_S = y_{S^j}+\cdots+y_{S^k}=c_j$$

if $j \le k$, and

$$\sum_{S:j\in S} y_S = 0 \ge c_j$$

if $j > k$. The dual value is

$$\sum_{j=1}^{k}(c_j - c_{j+1})f(S^j) + c_k f(S^k) = \sum_{j=1}^{k} c_j \left(f(S^j) - f(S^{j-1})\right),$$

which equals the primal value. Because of strong duality, both solutions are optimal.

9.40 Let E' be a subset of edges and let $U(E')$ denote the set of endpoints of E'. Then obviously $U(E(S))$ doesn't contain the vertices that are not connected to any of the other vertices in S. These isolated vertices increase the right-hand side of the subtour elimination inequality, but they don't contribute to the left-hand side. Hence we can confine the subtour constraints to those sets that don't contain isolated vertices,

$$\sum_{e\in E'=E(S)} x_e \le |U(E')| - 1 .$$

Even in this form many of the constraints are too loose to be relevant. Consider a set of edges E' that represents $c(E')$ connected components of the graph, say $E' = E'_1 \cup E'_2 \dots E'_c$. Then each $\mathbf{x} \in P_{\text{forest}}$ satisfies

$$\sum_{e\in E'_i} x_e \le |U(E'_i)| - 1 \quad \text{for } i = 1, \dots, c .$$

Summing these inequalities gives

$$\sum_{e\in E'} x_e \le \sum_{i=1}^{c}(|U(E'_i)| - 1) = |U(E')| - c(E') .$$

But $|U(E')| - c(E') = |V| - (|V| - |U|) - c(E') = r(E')$ since all the isolated vertices in $G(V, E')$ count as components.

The integrality of P_{forest} then follows from that fact that the rank is submodular and nondecreasing (Problem 9.38) and from the fact that submodular polytopes are integral (Problem 9.39).

The spanning tree polytope can be written as

$$P_{\text{tree}}^{\text{sub}} = P_{\text{forest}} \cap \left\{ \mathbf{x} : \sum_{e\in E} x_e = |V| - 1 \right\} .$$

But the cardinality constraint $\sum_{e\in E} x_e = |V| - 1$ is tangential to P_{forest} because the latter contains the constraint $\sum_{e\in E(V)} x_e \le |V| - 1$. Thus $P_{\text{tree}}^{\text{sub}}$ also has integral vertices.

9.41 Since f is unimodular and nondecreasing, for any $0 \le t < K$ we have

$$\begin{aligned} f(S_{\text{opt}}) &\le f(S_{\text{opt}} \cup S_t) \\ &\le f(S_t) + \sum_{e \in S_{\text{opt}} - S_t} \delta_e(S_t) \\ &\le f(S_t) + \left| S_{\text{opt}} - S_t \right| \eta_t \\ &\le f(S_t) + K\eta_t \\ &= \sum_{i=0}^{t-1} \eta_i + K\eta_t \,, \end{aligned}$$

proving (9.77). If $K^\star < K$, we have $\eta_{K^\star} = 0$. Then setting $t = K^\star$ gives

$$f(S_{\text{opt}}) = f(S_t) = f(S_{K^\star}) \,,$$

and the greedy algorithm is optimal.

Now we assume that $K^\star = K$, and determine the η_i where all K bounds (9.77) intersect, i.e., where the right-hand side is the same for all t. Setting the right-hand sides equal for t and $t+1$ gives

$$K\eta_t = \eta_t + K\eta_{t+1} \quad \text{or} \quad \eta_{t+1} = \left(1 - \frac{1}{K}\right)\eta_t \,.$$

We conclude that the η_i form a geometric series,

$$\eta_i = \left(1 - \frac{1}{K}\right)^i \eta_0 \,,$$

in which case $f(S_K)$ is the sum of the first K terms,

$$f(S_K) = \sum_{i=0}^{K-1} \eta_i = K\eta_0 \left(1 - \left(1 - \frac{1}{K}\right)^K\right)$$

while setting $t = 0$ in (9.77) gives

$$f(S_{\text{opt}}) \le K\eta_0 \,.$$

Comparing these gives

$$\frac{f(S_K)}{f(S_{\text{opt}})} \ge 1 - \left(1 - \frac{1}{K}\right)^K = \rho \,.$$

Finally, since $1 - x \le \mathrm{e}^{-x}$ we have $(1 - 1/K)^K \le 1/\mathrm{e}$, so $\rho \ge 1 - 1/\mathrm{e}$.

9.42 Any finite cut that separates s from t has to cut some links connected to t, and some links connected to s. The links connected to t represent the original set of vertices. If we cut a subset $S \subset V$ of these links, this will contribute $|S|$ to weight of the cut. To complete the cut, we have to separate s from all $e \notin E(S)$. Adding the weight of this part of the cut, we get

$$|S| + \sum_{e \in E} x_e - \sum_{e \in E(S)} x_e$$

for the total weight of the cut. Our separation oracle simply solves MIN CUT for this graph. If the minimum cut is lighter than $|V|$, it returns the violated subtour constraint corresponding to S.

However, the cut for $S = \emptyset$ has weight $|V| - 1$, so we have to exclude this solution. We do this by setting one of the weights x_e to infinity, thus forcing the two endpoints of edge e to be in the cut. If the resulting minimum cut is lighter than $|V|$, it yields a violated subtour constraint. If not, we set x_e back to its original value and put an infinite weight on another $x_{e'}$ that is part of the first cut, and repeat the procedure. This way we have check the minimum cut among all cuts with $|S| \geq 1$.

9.43 In (9.54b), the cut constraints for S and $V - S$ are redundant. Hence it is sufficient to keep only those constraints for $S \not\ni v_1$. Singling out $v_1 \in V$ and replacing the cut constraints by subtour constraints provides us with

$$\begin{aligned}
\sum_{e\in\delta(v_1)} x_e &= 2 \\
\sum_{e\in\delta(v)} x_e &= 2 && \text{for all } v \in V - v_1 \\
\sum_{e\in E(V-v_1)} x_e &= |V| - 2 \\
\sum_{e\in E(S)} x_e &= |S| - 1 && \text{for all } S \subset V,\, S \neq \emptyset \text{ and } v_1 \notin S \\
x_e &\leq 1 && \text{for } e \text{ incident to } v_1 \\
x_e &\geq 0
\end{aligned}$$

Note that the constraints $x_e \leq 1$ for e not incident to v_1 are part of the subtour elimination constraints.

Now we introduce dual variables π_v for the degree constraints for $v \in V - v_1$. Since the degree constraints are equality constraints, the dual variables π_v are unconstrained. We obtain the dual problem by removing the degree constraints, adding the term

$$\sum_{v\neq v_1} \pi_v \left(\sum_{e\in\delta(v)} x_e - 2 \right)$$

to the value function and maximizing with respect to π. If u and w denote the endpoints of edge e, we get

$$\max_{\pi} \left(-2 \sum_{v\neq v_1} \pi_v + \min_{\mathbf{x}} \left(\sum_e (c_e + \pi_u + \pi_w) x_e \right) \right)$$

subject to

$$
\begin{aligned}
\sum_{e\in\delta(v_1)} x_e &= 2 \\
x_e &\le 1 && \text{for } e \text{ incident to } v_1 \\
\sum_{e\in E(V-v_1)} x_e &= |V|-2 \\
\sum_{e\in E(S)} x_e &= |S|-1 && \text{for all } S\subset V,\ S\neq\emptyset \text{ and } v_1\notin S \\
x_e &\ge 0
\end{aligned}
$$

According to the first two constraints, it is obviously best to choose the two lightest edges incident to v_1. The other three constraints then specify a minimum spanning tree on $V - v_1$, compare to (9.47). Both optimizations refer to the modified edge weights $c_e + \pi_u + \pi_w$. Maximizing over π gives the Held–Karp bound (9.53), which by strong duality, equals the value of the LP relaxation of TSP.

Chapter 10

Randomized Algorithms

10.1 Let a_i denote the ith element in the list. There are 2^{n-1} orders in which, for each i, a_i is either the largest or the smallest element among $a_i, a_{i+1}, \ldots, a_n$. For instance, for $n = 5$ one such order is 5, 1, 4, 2, 3.

10.2 We give the solution using Jensen's inequality. Since the sum over all leaves of the probability that the walker arrives there is 1, we have

$$1 = \sum 2^{-d} = \ell\, \mathbb{E}[2^{-d}] \geq \ell\, 2^{-\mathbb{E}[d]} .$$

Multiplying both sides by $2^{-\mathbb{E}[d]}$ and taking the log of both sides gives the stated result.

10.3 Since the pivot is chosen randomly, each of the $j - i + 1$ items in the range $\{i, \ldots, j\}$ is equally likely to be chosen first as the pivot, and so

$$\mathbb{E}[X_{ij}] = \frac{2}{j - i + 1} .$$

Thus we have

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i<j} \frac{2}{j-i+1} = 2\sum_{j=1}^{n}\sum_{k=2}^{j} \frac{1}{k} \\ &= 2\sum_{j=1}^{n} (H_j - 1) \\ &\approx 2\int_1^n \ln x \,\mathrm{d}x = 2(n \ln n - n + 1) \\ &\approx 2n \ln n .\end{aligned}$$

10.4 After choosing a random pivot, the expected size of the sublist containing x is at most 3/4 times its former size, where this factor of 3/4 is achieved when x is in the center of a long list. Therefore the expected size of this sublist at depth d is at most $(3/4)^d n$. By the first moment method, the probability that x has not yet been chosen at depth $d = b \log n$ is at most this expectation, which is $n^{1-b\log(4/3)}$. By the union bound, the probability that not all x have been chosen at this depth is at most n times that, or $n^{2-b\log(4/3)}$. Setting $b = (2+a)/\log(4/3)$ makes this probability at most n^{-a}.

10.6 Let there be n vertices v_i with $i = 1, \ldots, n$, with two edges from s to each v_i and one edge from each v_i to t. The minimum cut consists of the n edges from the v_i to t. The algorithm finds this cut if and only if it merges all the v_i with s, and it does this with probability $(2/3)^n$.

10.7 If we arrive at $d = 0$ after $3d$ steps, we must have taken $2d$ steps to the left and d steps to the right. There are $\binom{3d}{d}$ ways to do this, and the probability of each one is $(1/3)^{2d}(2/3)^d$. Thus the total probability is

$$P = \binom{3d}{d}\left(\frac{1}{3}\right)^{2d}\left(\frac{2}{3}\right)^{d} = \frac{(3d)!}{d!(2d)!}\frac{2^d}{3^{3d}}\,.$$

Applying Stirling's approximation to the factorials gives

$$\begin{aligned} P &\approx \frac{(3d)^{3d}\,\mathrm{e}^{-3d}\sqrt{6\pi d}}{(2d)^{2d}\,\mathrm{e}^{-2d}\sqrt{4\pi d}\cdot d^d\,\mathrm{e}^{-d}\sqrt{2\pi d}}\,\frac{2^d}{3^{3d}} \\ &= \sqrt{\frac{3}{4\pi d}}\,\frac{3^{3d}}{2^{2d}}\,\frac{2^d}{3^{3d}} = \sqrt{\frac{3}{4\pi d}}\,\frac{1}{2^d} = \Theta(2^{-d}/\sqrt{d})\,. \end{aligned}$$

10.8 When we increase d by 1, the summand $\binom{n}{d}(k-1)^{-d}$ changes by a factor of

$$\frac{(n-d)}{(d+1)(k-1)}\,.$$

The summand is maximized when this ratio crosses 1. This occurs when $d = (n-k+1)/k$, which approaches n/k when n is large. Thus for the balanced case $k = 2$, the dominant contribution comes from the typical initial assignments for which $d = n/2$, while for $k = 3$, the dominant contribution comes from "exponentially lucky" initial assignments at $d = n/3$.

10.9 Analogous to k-SAT, the worst case occurs when exactly one of the endpoints of an edge agrees with C. Half the time, we change the color of that vertex, increasing the Hamming distance by 1. The other half of the time, we change the color of the other vertex to one of the other two colors. With probability 1/2, this new color agrees with C, decreasing the Hamming distance by 1; otherwise it still disagrees, and the Hamming distance remains the same. This gives us a biased random walk in which

$$\Pr[\Delta d = -1] = \frac{1}{4}\,,\quad \Pr[\Delta d = 0] = \frac{1}{4}\,,\quad \Pr[\Delta d = +1] = \frac{1}{2}\,.$$

The probability $p(d)$ we will reach the origin from an initial distance d then obeys the equation

$$p(d) = \frac{1}{4}p(d-1) + \frac{1}{4}p(d) + \frac{1}{2}p(d+1)\,,$$

giving $p(d) = 2^{-d}$ as before. There are $2^d\binom{n}{d}$ initial colorings whose Hamming distance from C is d, since each differing vertex can take either of the other two colors, and each initial coloring occurs with probability 3^{-n}. The total probability of success is then

$$P_{\text{success}} = 3^{-n}\sum_{d=0}^{n} 2^d\binom{n}{d}\,p(d) = 3^{-n}\sum_{d=0}^{n}\binom{n}{d} = (2/3)^n\,,$$

so the expected number of attempts is $(3/2)^n$.

10.10 For the reduction to 2-SAT we refer the reader to the solution for Problem 5.6. Assume that G is 3-colorable, and let C be a particular 3-coloring. If we forbid a random color at each vertex, the probability that it is consistent with C is $(2/3)^n$. The average number of restrictions we need to try before we succeed is thus at most $(3/2)^n$. Multiplying this by the polynomial running time of our favorite algorithm for 2-SAT gives the stated result.

10.11 In the exponential case, the probability distribution of the waiting time $t' = t - t_0$ is

$$\frac{P(t'+t_0)}{\int_{t_0}^{\infty} P(t)\,\mathrm{d}t} = \frac{(1/\tau)\,\mathrm{e}^{-(t'+t_0)/\tau}}{\mathrm{e}^{-t_0/\tau}} = P(t').$$

In the power-law case, after t_0 the expected time we have to wait is

$$\frac{\int_{t_0}^{\infty} t^{-\alpha}(t-t_0)\,\mathrm{d}t}{\int_{t_0}^{\infty} t^{-\alpha}\,\mathrm{d}t} = \frac{\int_{t_0}^{\infty} t^{1-\alpha}\,\mathrm{d}t}{\int_{t_0}^{\infty} t^{-\alpha}\,\mathrm{d}t} - t_0 = \frac{t_0}{\alpha-2},$$

which grows linearly with t_0.

10.12 Since the ratio between the algorithm's solution and the optimal one is at most 1, the expected ratio is bounded by

$$\alpha \le p(\varepsilon) + (1-p(\varepsilon))(\alpha-\varepsilon).$$

Solving for $p(\varepsilon)$ gives

$$p(\varepsilon) \ge \frac{\varepsilon}{1-(\alpha-\varepsilon)} \ge \varepsilon.$$

The expected number of attempts we need to achieve this ratio is then $1/p(\varepsilon) \le 1/\varepsilon$, and the probability that none of these attempts succeed is $(1-p(\varepsilon))^{1/p(\varepsilon)} < \mathrm{e}$.

10.13 There are $n(n/k-1)/2$ edges between vertices in the same group, for which $\mathbf{s}_i^T\mathbf{s}_j = 1$, and $n(n - n/k)/2$ edges between vertices in different groups, for which $\mathbf{s}_i^T\mathbf{s}_j = -1/(k-1)$. Thus the total weight of this vector cut is

$$W_{\text{VECTOR}} = \sum_{i<j} w_{ij}\,\frac{1-\mathbf{s}_i^T\mathbf{s}_j}{2} = \left(\frac{n(n-n/k)}{2}\right)\left(\frac{1+1/(k-1)}{2}\right) = \frac{n^2}{4}.$$

10.18 Consider a tree of depth 2, consisting of three NAND gates and four leaves. In order to determine the value of the root, we need to know the value of at least two of the leaves: if a leaf is `false` its parent is `true` and we need to know the value of the other parent, and similarly if a leaf is `true` we need to know the value of its sibling.

Conversely, if two siblings are `true` then their parent is `false` and the root is `true`, while if two cousins are `false`, their parents are `true` and the root is `false`. Since exactly one of these two cases holds, we can always determine the value of the root with the values of two of the leaves. Thus the size of the smallest witness doubles each time the depth increases by 2, which implies that it is $2^{k/2} = \sqrt{N}$.

10.19 If each child is `true` with probability p, their parent is `true` with probability

$$p' = 1 - p^2 .$$

Setting $p' = p$ gives the quadratic equation $p^2 + p - 1 = 0$, whose only positive root is $p = \varphi - 1$. Now, if we evaluate the children in random order, we only need to look at the second one if the first one is `true`. Thus the expected number of children we need to evaluate is $1 + p = \varphi$. In a tree of depth k, the expected number of leaves is then $\varphi^k = N^{\log_2 \varphi}$ as stated.

10.20 If a node is `false`, we have to evaluate all of its children. If it is `true`, we can stop as soon as we find a single `false` child. The worst case is when exactly one child is false. In that case, the expected number of `true` children we evaluate is $(d-1)/2$. This gives a pair of coupled equations which generalize (10.10) and (10.11):

$$\begin{aligned} f_{\texttt{true}}(k) &= f_{\texttt{false}}(k-1) + \frac{d-1}{2} f_{\texttt{true}}(k-1) \\ f_{\texttt{false}}(k) &= d\, f_{\texttt{true}}(k-1) . \end{aligned}$$

Combining these into a single equation gives a generalization of (10.12):

$$f_{\texttt{true}}(k) = \frac{d-1}{2} f_{\texttt{true}}(k-1) + d\, f_{\texttt{true}}(k-2) .$$

Then $f_{\texttt{true}}(k) = \Theta(\alpha^k)$ where α is the unique positive root of the equation

$$\alpha^2 - \frac{d-1}{2}\alpha - d = 0 .$$

This gives the stated value of α, and rest of the analysis proceeds as before.

10.22 Since the ith prime is at least i, if x has d distinct factors then $x \ge d!$. Stirling's approximation gives $d! \ge d^d \mathrm{e}^{-d}$, so if we pessimistically assume that $x = d!$ we have $\log x = \Theta(d \log d)$ and $\log\log x = \Theta(\log d)$. Then

$$d = \Theta\left(\frac{\log x}{\log d}\right) = \Theta\left(\frac{\log x}{\log\log x}\right) ,$$

Replacing Θ with O gives an upper bound that holds when $x \ge d!$.

10.23 To write x in this way, just let $z_p = 0$ or 1 if the number of factors of p in x's prime factorization is odd or even respectively. Then $s^2 = x / \prod p^{z_p}$ is a square because it contains an even number of factors of each p.

Now suppose $x \le t$. Since $s \le \sqrt{t}$, and since each z_p is either 0 or 1, the right-hand side of (10.30) can take at most $\sqrt{t}\, 2^{\pi(t)}$ different values. Since we have to be able to represent all t integers less than or equal to t, we have $t \le \sqrt{t}\, 2^{\pi(t)}$, and solving for $\pi(t)$ gives the stated bound.

10.24 Taking logs and using $\ln(1-x) \approx -x$ gives

$$\ln P(t) = \sum_{\text{prime } p < t} \ln\left(1 - \frac{1}{p}\right) \approx - \sum_{\text{prime } p < t} \frac{1}{p} .$$

Since each x contributes $1/x$ to this sum if x is prime, we write

$$\ln P(t) \approx -\sum_{x<t} \frac{P(x)}{x}.$$

When we increment t the sum on the right gains the term $P(t)/t$, so

$$\frac{\mathrm{d} \ln P(t)}{\mathrm{d}t} \approx P(t) - P(t-1) \approx -\frac{P(t)}{t}.$$

The solution to this differential equation is $P(t) = 1/(C + \ln t)$ for a constant C, since

$$\frac{\mathrm{d} \ln P(t)}{\mathrm{d}t} = -\frac{\mathrm{d}}{\mathrm{d}t} \ln(C + \ln t) = -\frac{1}{t(C + \ln t)} = -\frac{P(t)}{t}.$$

Thus $P(t) \approx 1/\ln t$ when t is large.

To approximate the integral of $P(t)$, note that

$$\frac{\mathrm{d}}{\mathrm{d}x} \frac{x}{\ln x} = \frac{1}{\ln x} - \frac{x}{\ln^2 x}.$$

Thus

$$\int_2^t \frac{\mathrm{d}x}{\ln x} = \frac{x}{\ln x}\Big|_2^t + \int_2^t \frac{x}{\ln^2 x}\,\mathrm{d}x = \frac{t}{\ln t} + O\left(\frac{t}{\ln^2 t}\right).$$

10.28 Setting the probability that x is prime to $P(x) = 1/\ln x$, the "expected" number of Mersenne primes is

$$\sum_{n=1}^{\infty} \frac{1}{\ln(2^n - 1)} \approx \frac{1}{\ln 2} \sum_{n=1}^{\infty} \frac{1}{n}.$$

This is the harmonic series, which diverges. On the other hand, the expected number of Fermat primes is

$$\sum_{k=0}^{\infty} \frac{1}{\ln(2^{2^k} + 1)} \approx \frac{1}{\ln 2} \sum_{k=0}^{\infty} \frac{1}{2^k},$$

giving a convergent geometric sum.

10.31 To save ink, we write $=$ and $\neq$ instead of $\equiv_p$ and $\not\equiv_p$. Suppose $\mathbf{AB} \neq \mathbf{C}$. Then $\mathbf{M} = \mathbf{AB} - \mathbf{C}$ is not the zero matrix. Choose some row $\mathbf{r}_i$ of $\mathbf{M}$ which is not all zeros. Then the ith component of $\mathbf{Mv}$ is the inner product $\mathbf{r}_i^T\mathbf{v}$. Suppose p is prime. For any vector $\mathbf{r}_i$ which is not all zeros, if $\mathbf{v}$ is random then $\mathbf{r}_i^T\mathbf{v}$ is uniformly random in $\mathbb{F}_p$, and the probability that it is 0 is $1/p$. Thus the probability that $\mathbf{Mv} = 0$, or equivalently that $\mathbf{A}(\mathbf{Bv}) = \mathbf{Cv}$, is at most $1/p$. If p is not prime, this probability is at most $1/q$ where q is the smallest prime factor of p.

10.34 (Partial solution) If $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ lie on a line, then if h sends any two of them to zero, it must send the third one to zero as well. The case $p = 2$ is special because no set of three points in the hypercube $\mathbb{F}_2^n$ are colinear. However, four points can lie in a plane, and if any three of them are sent to zero, so is the fourth. Thus h is not 4-wise independent.

10.37 If $\phi(x_1,\dots,x_n)=0$, either ψ is identically zero or it isn't. If it is, then in particular the coefficient of x_n^i in ψ, namely $\phi_i(x_1,\dots,x_{n-1})$, must be zero. By induction, the probability of that is at most $(d-i)/|S|$ since the total degree of ϕ_i is at most $d-i$. If ψ is not identically zero, the probability that $\psi(x_n)=0$ is at most $i/|S|$ since ψ has degree i. Combining these two probabilities gives $d/|S|$ as stated.

10.41 Since $q\geq 2$ we have $k\leq \log_2 p=O(n)$. Thus there are only $O(n)$ values of k we need to try. For each one we can do binary search on q, comparing q^k to p, until we either find q or prove that $p^{1/k}$ is not an integer. This takes a total of $O(n^2)$ calls to MODULAR EXPONENTIATION, so this problem is in P. If we like, we can then check where q is prime in polynomial time.

10.45 Let N_k denote the number of steps where $r<2^{-k}$. On any such step, if the counter is k or less, we increment it, so if $N_k\geq k$ the final value of the counter is at least k. On the other hand, if $N_k=0$ then the counter never advances beyond k.

Now, N_k is a binomial random variable $\mathrm{Bin}(N,2^{-k})$ and has expectation is $2^{-k}N$. If $k=(1-\varepsilon)\log_2 N$, this expectation is N^ε. By Theorem A.2, the probability that $N_k<k<N^\varepsilon/2$ is at most $\mathrm{e}^{-N^\varepsilon/8}=o(1)$. Thus with high probability we have $N_k\geq k$, and the final value of the counter is at least $(1-\varepsilon)\log_2 N$.

On the other hand, if $k=(1+\varepsilon)\log_2 N$ the expectation of N_k is $N^{-\varepsilon}$. The probability that $N_k\geq 1$ is then at most $N^{-\varepsilon}=o(1)$ by Markov's inequality or equivalently by the union bound. Thus with high probability we have $N_k=0$ in this case, and the final value of the counter is at most $(1+\varepsilon)\log_2 N$.

10.47 We can rewrite write $p_{A,B}(h)$ as

$$p_{A,B}(h)=\frac{1}{2^\ell}\sum_{x\in A}\mathbb{1}[h(x)\in B]\,,$$

where $\mathbb{1}[h(x)\in B]$ is the indicator random variable which is 1 if $h(x)\in B$ and 0 if $h(x)\notin B$. By linearity of expectation and the definition (10.20) of pairwise independence, we have

$$\mathop{\mathbb{E}}_h p_{A,B}(h)=\frac{1}{2^\ell}\sum_{x\in A}\Pr_h[h(x)\in B]=\frac{|A|}{2^\ell}\frac{|B|}{2^\ell}=\alpha\beta\,.$$

By pairwise independence, for any distinct $x,y\in A$ we have

$$\Pr_h[h(x)\in B \text{ and } h(y)\in B]=\left(\frac{B}{2^\ell}\right)^2=\beta^2\,,$$

so the second moment of $p_{A,B}$ is

$$\begin{aligned}\mathop{\mathbb{E}}_h p_{A,B}(h)^2&=\frac{1}{2^{2\ell}}\sum_{x,y\in A}\Pr_h[h(x)\in B \text{ and } h(y)\in B]\\&=\frac{|A|\beta+|A|(|A|-1)\beta^2}{2^{2\ell}}\\&=\frac{\alpha\beta(1-\beta)}{2^\ell}+\alpha^2\beta^2\,.\end{aligned}$$

Combining these gives the variance,

$$\mathrm{Var}_h\, p_{A,B}(h) = \frac{\alpha\beta(1-\beta)}{2^\ell} \le \frac{\alpha\beta}{2^\ell}\,.$$

Note what's going on here: due to pairwise independence, the bulk of "off-diagonal" terms of the second moment where $x \neq y$ exactly cancel the square of the expectation, leaving the "diagonal" terms with $x = y$ to make up the variance. This is the same variance we would have if every function value $h(x)$ were independently uniform in $\{0,1\}^\ell$.

Finally, Chebyshev's inequality (A.15) gives

$$\Pr_h\left[\left|p_{A,B}(h) - \alpha\beta\right| > \varepsilon\right] \le \frac{1}{\varepsilon^2}\,\mathrm{Var}_h\, p_{A,B}(h) \le \frac{\alpha\beta}{2^\ell \varepsilon^2}\,.$$

Chapter 11

Interaction and Pseudorandomness

11.2 In order for two independent samples $(s, H(s,x))$ and $(s', H(s',x'))$ of P to be the same, we need $s = s'$ and $H(s,x) = H(s,x')$. Since s is chosen uniformly, the probability that $s = s'$ is 2^{-k}. The probability that $H(s,x) = H(s,x')$ is at most the probability that $x = x'$, which is less than or equal to 2^{-b}, plus the probability that $x \neq x'$ but $h_s(x) = h_s(x')$, which is $2^{-\ell}$ by pairwise independence. Thus

$$\|P\|_2^2 \le 2^{-k}\left(2^{-\ell} + 2^{-b}\right) = 2^{-n'} + 2^{-(k+b)}$$

If U is the uniform distribution on $\{0,1\}^{n'}$, then $\|U\|_2^2 = P^T U = 2^{-n'}$. So

$$\|P - U\|_2^2 = \|P\|_2^2 + \|U\|_2^2 - 2P^T U = 2^{-(k+b)},$$

and Cauchy–Schwarz gives

$$\|P - U\|_1 \le 2^{n'/2}\,\|P - U\|_2 \le 2^{-(b-\ell)/2} = \varepsilon\,.$$

11.6 For an interactive proof of Quadratic Nonresidue with private coins, Arthur chooses a random y and shows Merlin y^2 and xy^2 in random order—analogous to showing Merlin a randomly permuted graph. He then challenges Merlin to tell him which is which. If x is a nonresidue, then y^2 is a residue but xy^2 is not, so Merlin can win every time. Bu if x is a residue, y^2 and xy^2 are both uniformly random residues, so Merlin can only win with probability 1/2.

To do this with public coins, let R denote the subgroup of residues, and let Merlin prove $|R|$ to Arthur. If we define $S = R \cup xR$, Merlin can prove that a given $z \in S$ by providing a y such that $z = y^2$ or $z = xy^2$. We have $|S| = |R|$ if x is a residue, and is $2|R|$ if it is not. We can amplify this factor of 2 to 16 by taking $S' = S^4$, and Merlin can then prove interactively that $|S| = 2|R|$ as in Section 11.1.2.

11.7 For a zero-knowledge proof of Quadratic Residue, Merlin chooses a random y and shows y^2 to Arthur. If x is a residue then xy^2 is also. So, Arthur is allowed to demand either y or $z = y\sqrt{x}$ such that $z^2 = xy^2$—analogous to asking for an isomorphism from H to G_1 or G_2. Since y is random, z is too, so in either case Arthur learns nothing.

11.8 If $\ell = x/(p-1)$, the first binary digit of ℓ is $b(x)$. We can determine successive digits of ℓ as follows. First, if $\ell \ge 1/2$, we subtract 1/2 from it by multiplying z by $a^{-(p-1)/2}$, which equals $a^{(p-1)/2}$ by Fermat's

Little Theorem. We then double ℓ by squaring z, and again check whether $\ell \geq 1/2$. After finding $\log p = O(n)$ binary digits, we have determined $x = \ell(p-1)$ to within the nearest integer.

For instance, in the example where $p = 13$ and $z = 9$, we have $x = 8$ so ℓ's first digit is $b(x) = 1$. Multiplying z by $2^6 = 12$ gives 4, and squaring the result gives $z = 4^2 = 3$. Now $x = 4$, so the next digit is $b(x) = 0$. Squaring $z = 3$ gives $z = 9$ again. In this case, ℓ's digits alternate between 1 and 0, so $\ell = 2/3$ and $\log_2 9 = 8$.

If we can guess $b(x)$ correctly with probability $1/2 + 1/n^c$ or more, the majority of n^{c+1} trials gives $b(x)$ with probability exponentially close to 1. Since we need to find $O(n)$ digits, with high probability all of these are correct.

11.9 If $x = \log_a z$ is even, then $z = y^2$ where $y = a^{x/2}$. Fermat's Little Theorem states that $y^{p-1} = 1$, so this implies that $z^{(p-1)/2} = 1$. Conversely, if x is odd then $z^{(p-1)/2} = a^{(p-1)/2} = -1$ since ± 1 are the only square roots of 1 mod a prime. We can determine $z^{(p-1)/2}$ mod p in polynomial time. If a is a primitive root then x is a quadratic residue if and only if $\log_a x$ is even, so QUADRATIC RESIDUE is in P in the case where the modulus is a prime.

11.10 Choose y uniformly at random from $\mathbb{Z}_{p-1} = \{0, \ldots, p-2\}$, and let $w = a^y z$. Then w is uniformly random in $\mathbb{Z}_p^*$, so $w \in S$ with probability $|S| / \left|\mathbb{Z}_p^*\right|$. Finally, $\log_a z = (\log_a w) - y$, where we do this subtraction mod $p-1$.

11.11 Since $M(x)$ has degree $n-1$ and each term in $\operatorname{perm} M(x)$ is the product of $n-1$ entries of $M(x)$, the degree of $Q(x)$ is at most $(n-1)^2 < n^2$. Thus if Arthur chooses x from the range $0 \leq x < n^4$, say, he will catch Merlin with probability $1 - 1/n^2$ if Merlin gives him the wrong $Q(x)$. Taking the union bound over all n stages of this process, the probability that Merlin fools Arthur is at most $1/n$.

11.18 We'll skip some of the easy parts. But we have $M^2 - N^2 = M(M-N) + (M-N)N$, so if $\|M\| = \|N\| = 1$ then

$$\left\|M^2 - N^2\right\| \leq \|M(M-N)\| + \|(M-N)N\| \leq \|M-N\| (\|M\| + \|N\|) = 2\|M-N\| .$$

We get (11.39) from Problem 10.47 by setting $\varepsilon = \delta/m^2$, so that even after summing over all k to compute the square, and over all j to sum an entire row, we still have $\left\|N^{(2)} - (N^{(1)})^2\right\| \leq \delta$. We have m^3 triples i, j, k, so taking the union bound multiplies by another factor of m^3, bringing the total probability of failure to $m^3/(2^\ell \varepsilon^2) = m^7/(2^\ell \delta^2)$.

Combining (11.38) and (11.39), or rather its generalization to all b, with the triangle inequality gives the recurrence

$$\left\|M^{(2b)} - N^{(2b)}\right\| \leq \left\|M^{(2b)} - (N^{(b)})^2\right\| + \left\|(N^{(b)})^2 - N^{(2b)}\right\| \leq 2\left\|M^{(b)} - N^{(b)}\right\| + \delta .$$

This is the Towers of Hanoi recurrence (3.1), and iterating it $z = \log_2 b$ times gives

$$\left\|M^{(2b)} - N^{(2b)}\right\| \leq (2^z - 1)\delta = (b-1)\delta .$$

Since $b \leq t \leq 2^s$ and $m = 2^s$, for any constant δ we can make the probability of failure tend to zero (indeed, exponentially fast as a function of s) by setting $\ell = 10s$.

Chapter 12

Random Walks and Rapid Mixing

12.3 We have

$$\frac{\partial S}{\partial p(x)} = -\ln p(x) - 1\,, \ \frac{\partial P}{\partial p(x)} = 1\,, \ \text{and} \ \frac{\partial E}{\partial p(x)} = E(x)\,.$$

Using Lagrange multipliers α and β then gives

$$-\ln p(x) - 1 = \alpha + \beta E(x)\,,$$

or

$$p(x) = \mathrm{e}^{-(\alpha+1)}\,\mathrm{e}^{-\beta E(x)} = \mathrm{e}^{-\beta E(x)}/Z\,,$$

where $Z = \mathrm{e}^{\alpha+1}$.

12.30 Going around all three sides of the triangle increases or decreases the height by 3. Therefore, there must be a topological defect somewhere in the interior.

12.33 Suppose that v is an eigenvector of M with eigenvalue λ, i.e., $Mv = \lambda v$. Since M is stochastic, we have $\mathbf{1}^T M = \mathbf{1}^T$. Then

$$\mathbf{1}^T v = \mathbf{1}^T M v = \lambda \mathbf{1}^T v\,.$$

Therefore, if $\lambda \neq 1$, we must have $\mathbf{1}^T v = 0$.

If M is diagonalizable, its eigenvectors P_{eq} and v_k span the vector space, so we can write any vector w as a linear combination

$$w = a_{\text{eq}} P_{\text{eq}} + \sum_{k \neq 0} a_k v_k\,.$$

If w is a probability distribution, multiplying both sides on the left by $\mathbf{1}^T$ gives

$$1 = \mathbf{1}^T w = a_{\text{eq}}\,\mathbf{1}^T P_{\text{eq}} + \sum_{k \neq 0} a_k\,\mathbf{1}^T v_k = a_{\text{eq}}\,,$$

so $a_{\text{eq}} = 1$.

12.34 We have

$$M'_{xy} = T^{-1}_{xx}\, M_{xy}\, T_{yy} = \sqrt{\frac{P_{\rm eq}(y)}{P_{\rm eq}(x)}}\, M(y \to x)\,.$$

Detailed balance states that $P_{\rm eq}(x)\,M(x \to y) = P_{\rm eq}(y)\,M(y \to x)$, and dividing both sides by $\sqrt{P_{\rm eq}(x)\,P_{\rm eq}(y)}$ gives $M'_{yx} = M'_{xy}$.

12.39 For any particular v with $\sum_x v(x) = 0$, we have

$$\lambda_1 = 1 - \delta \geq \frac{v^T M v}{|v|^2}\,.$$

Now we have

$$|v|^2 = v^T v = \frac{1}{|S|} + \frac{1}{|\overline{S}|}\,.$$

Writing $v(S) = \sum_{x \in S} v(x)$ and so forth, we have

$$\begin{aligned} v^T M v &= \frac{1}{|S|}(Mv)(S) - \frac{1}{|\overline{S}|}(Mv)(\overline{S}) \\ &= \left(\frac{1}{|S|} + \frac{1}{|\overline{S}|}\right)(Mv)(S) \end{aligned}$$

since $(Mv)(S) + (Mv)(\overline{S}) = 0$.

At equilibrium, a fraction $\Phi(S)$ of the probability in S flows to $\overline{S}$, and a fraction $(|S|/|\overline{S}|)\Phi(S)$ of the probability in $\overline{S}$ flows to S. Thus

$$(Mv)(S) = (1 - \Phi(S))\, v(S) + \frac{|S|}{|\overline{S}|}\Phi(S)\, v(\overline{S}) = 1 - \Phi(S)\left(1 + \frac{|S|}{|\overline{S}|}\right) \geq 1 - 2\Phi(S)\,.$$

Combining these gives $\delta \leq 2\Phi(S)$, and minimizing over all S gives $\delta \leq 2\Phi$.

12.41 Since v is orthogonal to the uniform distribution, its 2-norm decreases by a factor of $1 - \delta$ on each step. By Cauchy–Schwarz,

$$\left\|M^t v\right\|_1 \leq \sqrt{N}\left\|M^t v\right\|_2 \leq \sqrt{N}(1-\delta)^t\, \|v\|_2 = \sqrt{\frac{N}{2}}\,(1-\delta)^t\,.$$

Demanding that this is less than 1 and setting $D = 2t$ gives

$$D \leq 2\left(\frac{\ln\sqrt{N/2}}{-\ln(1-\delta)} + 1\right) \leq \frac{\ln N}{-\ln(1-\delta)} + 2\,.$$

The simpler bound follows from the fact that $-\ln(1-\delta) \geq \delta$.

12.42 We have $M v_{\mathrm{eq}} = v_{\mathrm{eq}}$ and $\left\|M^t v_{\mathrm{neq}}\right\|_2 \le (1-\delta)^t \left\|v_{\mathrm{neq}}\right\|_2$. Then the triangle inequality gives

$$\begin{aligned} \left\|\Pi M^t v\right\|_2 &\le \left\|\Pi M^t v_{\mathrm{eq}}\right\|_2 + \left\|\Pi M_t v_{\mathrm{neq}}\right\|_2 \\ &\le \left\|\Pi v_{\mathrm{eq}}\right\|_2 + (1-\delta)^t \left\|v_{\mathrm{neq}}\right\|_2 , \end{aligned} \tag{a}$$

where we used the fact that multiplying a vector by Π can never increase its norm. To bound the first term, note that $\left\|v_{\mathrm{eq}}\right\|_1 = \|v\|_1$, so

$$v_{\mathrm{eq}} = \frac{\|v\|_1}{N}(1,\ldots,1) \text{ and } \Pi v_{\mathrm{eq}} = \frac{\|v\|_1}{N}(\underbrace{0,\ldots,0}_{W},\underbrace{1,\ldots,1}_{\overline{W}}).$$

Since v is supported only on $\overline{W}$, the Cauchy–Schwarz inequality implies that

$$\|v\|_1 \le \sqrt{\left|\overline{W}\right|}\,\|v\|_2 .$$

Using $|W| \ge N/2$, we then have

$$\left\|\Pi v_{\mathrm{eq}}\right\|_2 \le \frac{\|v\|_1}{N}\sqrt{\left|\overline{W}\right|} \le \frac{\|v\|_2}{N}\left|\overline{W}\right| \le \frac{1}{2}\|v\|_2 . \tag{b}$$

Finally, we have $\left\|v_{\mathrm{neq}}\right\|_2 \le \sqrt{\left\|v_{\mathrm{eq}}\right\|_2^2 + \left\|v_{\mathrm{neq}}\right\|_2^2} = \|v\|_2$. Combining this and (b) with (a) completes the proof.

12.48 Since there is one returning path of length $t = 0$, we can use the boundary condition $h(0) = 1$ to distinguish between the two roots of the quadratic equation for $h(z)$. This gives

$$\begin{aligned} g(z) &= \frac{2(d-1)}{d-2+d\sqrt{1-4(d-1)z^2}} \\ h(z) &= \frac{1-\sqrt{1-4(d-1)z^2}}{2(d-1)z^2} . \end{aligned}$$

Their radius of convergence is $1/(2\sqrt{d-1})$, since at this point $\sqrt{1-4(d-1)z^2}$ becomes complex.

Chapter 13

Counting, Sampling, and Statistical Physics

13.13 To produce G_k, attach each vertex G to a fan of $k-1$ additional vertices. Then if v is a hole in a matching of G, we can produce a matching of G_k by connecting v to one of these vertices, or none at all. Since we can make this choice independently for each hole, G_k has t^k matchings for each matching of G with t holes.

To reduce #MATCHINGS to #2-SAT, define a Boolean variable x_e for each edge. For each pair of edges e, e' that meet at a vertex, include the clause $(\overline{x}_e \vee \overline{x}_{e'})$ so that the matching can include at most one of them.

13.15 If the median of t trials is outside the range $[(1+\varepsilon)^{-1}A(x), (1+\varepsilon)A(x)]$, then at least $t/2$ of the trials fell outside this range. Since each trial does this with probability at most 1/3, the probability of this is at most

$$\sum_{j=t/2}^{t} \binom{t}{j} \left(\frac{1}{3}\right)^j \left(\frac{2}{3}\right)^{t-j} \le \frac{t}{2} \binom{t}{t/2} \left(\frac{2}{9}\right)^{t/2} \le \frac{t}{2} \left(\frac{8}{9}\right)^{t/2} = \mathrm{e}^{-\Omega(t)}.$$

Setting this equal to P_{fail} and inverting gives $t = O(\log P_{\text{fail}}^{-1})$.

13.20 For any integer matrix M, $\operatorname{perm} M \equiv_2 \det M$.

13.26 The hexagonal lattice is bipartite, hence all cycles have even length $2k$. We learned in Sec. 13.6.1 that we need to put weights $w_{ij} = \pm 1$ on the edges such that the product of weights of each face in a planar graph is -1 if k is even and $+1$ if k is odd to get $\operatorname{perm} A = |\det A'|$. Now in the hexagonal lattice, each face has $k = 3$, hence we get away by putting weight $+1$ on each edge, i.e. by not changing the adjacency matrix at all: $\operatorname{perm} A = |\det A|$.

We compute $\det A$ by treating the hexagonal lattice as $\ell \times \ell$ square lattice with two vertices in each cell, as shown in Figure 13.30, i.e. $n = 2\ell^2$. The eigenvectors are then given by Fourier basis vectors

$$v_{jk}(x,y) = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \mathrm{e}^{2\pi\imath(jx+ky)/\ell} \qquad 0 \le j,k < \ell\,.$$

Now A acts on this Eigenvektor for one cell throug the one internal link and the four links that connect to vertices in the cells left, right, below and above the current cell. With

$$\theta = \frac{2\pi j}{\ell} \qquad \phi = \frac{2\pi k}{\ell}$$

we get for the 2×2 matrix A'_{jk} that acts on the Eigenvector v_{ij}

$$A'_{jk} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & e^{\imath\theta} \\ e^{-\imath\theta} & 0 \end{pmatrix} + \begin{pmatrix} 0 & e^{\imath\phi} \\ e^{-\imath\phi} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1+e^{\imath\theta}+e^{\imath\phi} \\ 1+e^{-\imath\theta}+e^{-\imath\phi} & 0 \end{pmatrix}.$$

From $\det A'_{jk} = -(3+2\cos\theta+2\cos\phi+2\cos(\theta-\phi)$ we get

$$\begin{aligned} s &= \frac{1}{2}\lim_{\ell\to\infty}\frac{1}{2\ell^2}\ln|\prod_{jk}\det A'_{jk}| = \frac{1}{2}\lim_{\ell\to\infty}\frac{1}{2\ell^2}\sum_{jk}\ln|\det A'_{jk}| \\ &= \frac{1}{2}\lim_{\ell\to\infty}\frac{1}{2\ell^2}\sum_{jk}\ln\left[3+2\cos\frac{2\pi j}{\ell}+2\cos\frac{2\pi k}{\ell}+2\cos\left(\frac{2\pi j}{\ell}-\frac{2\pi k}{\ell}\right)\right] \\ &= \frac{1}{16\pi^2}\int_0^{2\pi}\int_0^{2\pi}\ln\left[3+2\cos\theta+2\cos\phi+2\cos(\theta-\phi)\right]\mathrm{d}\theta\,\mathrm{d}\phi\,. \end{aligned}$$

This integral can be simplified by expressing it with new variables

$$\omega = \frac{1}{2}(\theta-\phi) \qquad \alpha = \frac{1}{2}(\theta+\phi)$$

and using the identity $\cos(x\pm y) = \cos x\cos y \mp \sin x \sin y$. We get

$$s = \frac{1}{16\pi^2}\int_{-\pi}^{\pi}\mathrm{d}\omega\int_0^{2\pi}\mathrm{d}\alpha\,\ln\left[3+4\cos\alpha\cos\omega+2\cos 2\omega\right].$$

Now the integral over α can be done using the formula

$$\int_0^{2\pi}\ln(a+b\cos x)\,\mathrm{d}x = 2\pi\ln\frac{a+\sqrt{a^2-b^2}}{2} \qquad (a\geq|b|>0).$$

In our case $b = 4\cos\omega$ and

$$a = 3+2\cos 2\omega = 1+4\cos^2\omega\,,$$

and therefore

$$\frac{a+\sqrt{a^2-b^2}}{2} = \begin{cases} 1 & \cos^2\omega < \frac{1}{4} \\ 4\cos^2\omega & \cos^2\omega \geq \frac{1}{4}\,. \end{cases}$$

Hence doing the α integral provides us with

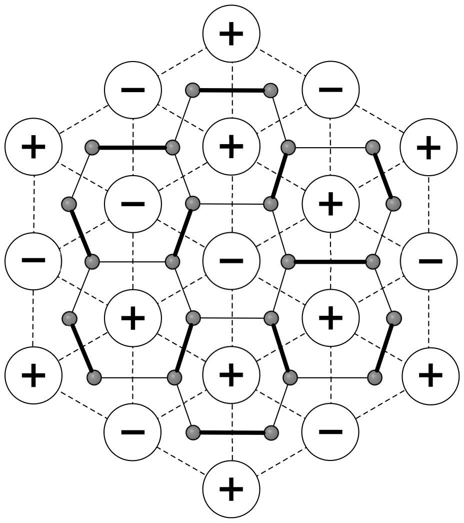
$$s = \frac{1}{\pi}\int_0^{\pi/3}\ln(2\cos\omega)\,\mathrm{d}\omega = \frac{1}{2\pi}\,\mathrm{Cl}_2(\pi/3),$$

where Cl_2 is the Clausen function of order 2,

$$\mathrm{Cl}_2(\phi) = -\int_0^{\phi} \ln\left|2\sin\frac{x}{2}\right| \mathrm{d}x = \sum_{k=1}^{\infty} \frac{\sin k\phi}{k^2}\,.$$

The numerical value is $s = 0.1615329736\ldots$.

13.27 The hexagonal lattice is the dual of the triangular lattice (see figure) The groundstate of the antiferromagnetic Ising model on the triangular lattice is a configuration where in each face there is exactly one unsatisfied bond that connects parallel spins. Match two vertices of the dual lattice if and only if the correspondig faces of the triangular lattice share an unsatisfied bond. This is a one-to-one correspondence between perfect matchings of the hexagonal lattice and groundstates of the antiferromagnet on



the triangular lattice.

13.34 We can write $P(G, p)$ recursively in the following way:

$$Q_p(G) = \begin{cases} pQ_p(G \cdot e) & \text{if } e \text{ is a bridge} \\ Q_p(G - e) & \text{if } e \text{ is a self-loop} \\ (1-p)Q_p(G-e) + pQ_p(G \cdot e) & \text{otherwise} \\ 1 & \text{if } G \text{ is a single vertex}. \end{cases}$$

In order to make $(1-p)Q_p(G-e)+pQ_p(G\cdot e)$ look like $T(G-e)+T(G\cdot e)$, we set $Q_p(G) = Ca^m b^n T(G; x, y)$ where $a = 1-p$ and $b = p/(1-p)$. The first two relations then determine $x = 1$ and $y = 1/(1-p)$, and the last relation determines $C = (1-p)/p$.

Alternately, we can write $Q_p(G)$ directly as a sum over all connected spanning subgraphs. By setting $x = 1$ in (13.52), we ensure that the only subgraphs which span contribute, i.e., those with $c(S) = c(G) = 1$.

Then summing the probability that the edges in S, and only these, are kept gives

$$\begin{aligned} Q_p(G) &= \sum_{\text{spanning } S} p^{|S|}(1-p)^{m-|S|} \\ &= p^{n-1}(1-p)^{m-n+1} \sum_{\text{spanning } S} (p/(1-p))^{1+|S|-n} \\ &= p^{n-1}(1-p)^{m-n+1}\, T(G;1,y), \end{aligned}$$

where $y-1 = p/(1-p)$.

13.35 First show that

$$Z(G) = \begin{cases} \mathrm{e}^{\beta} Z(G-e) & \text{if } e \text{ is a self-loop} \\ (\mathrm{e}^{\beta}+\mathrm{e}^{-\beta}) Z(G\cdot e) & \text{if } e \text{ is a bridge} \\ \mathrm{e}^{-\beta} Z(G-e) + (\mathrm{e}^{\beta}-\mathrm{e}^{-\beta}) Z(G\cdot e) & \text{otherwise} \\ 2 & \text{if } G \text{ is a single vertex}. \end{cases}$$

The first of these follows because adding a self-loop just increases the weight of any state by e^{β}. For the second and third, we have

$$Z = \mathrm{e}^{\beta} Z_{\text{same}} + \mathrm{e}^{-\beta} Z_{\text{different}}$$

where Z_{same} and $Z_{\text{different}}$ are the partition functions, assuming that e is absent, summed over the states where the spins of e's endpoints are the same or different. If e is a bridge then $Z_{\text{same}} = Z_{\text{different}} = Z(G\cdot e)$. If it is not, then

$$Z(G\cdot e) = Z_{\text{same}} \text{ and } Z(G-e) = Z_{\text{same}} + Z_{\text{different}}.$$

Trying a solution of the form $Z(G) = Ca^m b^n T(G;x,y)$ and fitting the above recursion, and then applying (13.52) and simplifying, gives the stated result.

13.37 Each vertex v is the root of a subtree. Let $Z(v) = Z_0(v) + Z_1(v)$ be the partition function of the hard sphere model on this subtree, where $Z_0(v)$ and $Z_1(v)$ sum over the independent sets S where $v \notin S$ and $v \in S$ respectively. Our goal is to compute $Z(r)$ for the root vertex r.

For each vertex v, let $C(v)$ denote the set of v's children. If $v \notin S$, then S consists of an independent set S_w on the subtree of each child $w \in C(v)$. Summing over all combinations of these subtrees gives the product of their partition functions,

$$Z_0(v) = \prod_{w\in C(v)} Z(w) = \prod_{w\in C(v)} \big(Z_0(w) + Z_1(w)\big).$$

If $v \in S$, we have to exclude v's children from S, so $w \notin S_w$ for all $w \in C(v)$. On the other hand, including v increases $|S|$ by one, multiplying the partition function by a factor of λ. Thus

$$Z_1(v) = \lambda \prod_{w\in C(v)} Z_0(w).$$

We apply these equations recursively, until we get to the base cases $Z_0(v) = 1$ and $Z_1(v) = \lambda$ if v is a leaf, i.e., if $C(v) = \emptyset$. Using dynamic programming, we only have to compute $Z_0(v)$ and $Z_1(v)$ once for each vertex v, so the total amount of work we have to do is polynomial (indeed, linear) in the number of vertices.

13.40 The transfer matrix is

$$M = \begin{pmatrix} e^{\beta(1+h)} & e^{-\beta(1-h)} \\ e^{-\beta(1+h)} & e^{\beta(1-h)} \end{pmatrix}$$

and the rest is arithmetic.

13.41 If we normalize the transfer matrix, we get a stochastic transition matrix $M' = M/\cosh\beta$. We can then write

$$\mathbb{E}[s_i s_j] = (1,0)\cdot(M')^\ell \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = (1,0)\cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tanh^\ell \beta = \tanh^\ell \beta\,,$$

since $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ is an eigenvalue of M' with eigenvector $\tanh\beta$. At low temperature, i.e., large β, we have $\xi \approx e^{2\beta}$. At high temperature, i.e., small β, we have $\xi \approx -1/\ln\beta$.

13.44 By calculating $Z(\beta)$ at $2m+1$ different values of β, we can learn the coefficient of $e^{-\beta E}$ for all $-m \le E \le m$. In particular, we can learn the number of states with the largest possible energy. But this is #MAX CUTS, which by Problem 13.11 is #P-complete.

Chapter 14

When Formulas Freeze: Phase Transitions in Computation

14.1 Let $P_{G(n,m)}(c)$ and $P_{G(n,p)}(c)$ denote the probabilities that P holds in each model. Since P is monotone decreasing, both of these are decreasing functions of c.

The number of edges in $G(n, p = c/n)$ is binomially distributed with mean $\mathbb{E}[m] = p\binom{n}{2} = cn/2 + o(n)$, and the Chernoff bound implies that m is within $o(n)$ of this mean with high probability. Therefore, for any $c' > c$ we can generate a uniformly random graph $G(n, m' = c'n/2)$ with high probability by starting with $G(n, p = c/n)$ and adding $m' - m$ random edges. Adding edges can only decrease the probability that P holds, so for any $c' > c$ we have

$$P_{G(n,m)}(c') < P_{G(n,p)}(c) + o(1).$$

Similarly, for any $c' < c$ we can generate $G(n, m' = c'n/2)$ with high probability by removing $m - m'$ random edges from $G(n, p = c/n)$. Removing edges can only increase the probability that P holds, so for any $c' < c$,

$$P_{G(n,m)}(c') > P_{G(n,p)}(c) - o(1).$$

These inequalities imply that $G(n, p)$ and $G(n, m)$ have the same threshold. For instance, suppose that $P_{G(n,p)}(c) = 1 - o(1)$ for all $c < c^*$. Then for any $c' < c^*$ we also have

$$P_{G(n,m)}(c') > P_{G(n,p)}(c) - o(1) = 1 - o(1),$$

since there is a c such that $c' < c < c^*$. The other cases follow similarly.

14.2 The number of ways that ℓ vertices can be connected in an undirected cycle is $(\ell - 1)!/2$. There are $\ell!$ orders in which we can list them, but it doesn't matter where we start the cycle or in which direction we go. Since the probability that all ℓ edges in a given cycle exist is p^ℓ, the expected number of cycles of length ℓ is then

$$\binom{n}{\ell} \frac{(\ell-1)!}{2} p^\ell \le \frac{n^\ell p^\ell}{2\ell} = \frac{c^\ell}{2\ell},$$

where we used the bound $\binom{n}{\ell} \le n^\ell/\ell!$. The expected number of vertices in such a cycle is ℓ times this, or $c^\ell/2$. Thus in the limit $n \to \infty$, the total number of vertices in a cycle of length $\ell \ge 3$ is a geometric series, which converges if $c < 1$:

$$\frac{1}{2}\sum_{\ell=3}^{\infty} c^\ell \le \frac{1}{2}\frac{1}{1-c} = O(1) .$$

For the second part, the expected number of vertices in cycles of length $B \log n$ or less is

$$\frac{1}{2}\sum_{\ell=3}^{B\log n} c^\ell .$$

If $c < 1$, this is $O(1)$. If $c = 1$, all these terms are the same size and the sum is $O(\log n)$. If $c > 1$, they grow geometrically, and the sum is a constant times its largest term. This is $c^{B\log n} = n^{B\log c}$, which is $o(n)$ for any $B < 1/\log c$.

14.3 $\mathbb{E}[T_k]$ is the number of ways to choose k vertices, times the number of ways to connect them together in a tree, times the probability that its $k-1$ edges exist, times the probability that no edges connect it to the rest of the graph. Ignoring whether or not the other internal edges of the component exists gives (14.90).

Using the bounds $1 - p \le \mathrm{e}^{-p}$ and $\binom{n}{k} \le n^k/k!$, Stirling's approximation $k! > \sqrt{2\pi k}\, k^k \mathrm{e}^{-k}$, and taking $k = o(\sqrt{n})$, (14.90) becomes

$$\begin{aligned}\mathbb{E}[T_k] &\le \binom{n}{k} k^{k-2} p^{k-1} (1-p)^{k(n-k)} \\ &\le \frac{n^k\, k^{k-2}\, c^{k-1}}{k!\, n^{k-1}}\, \mathrm{e}^{-ck} \left(1 + O(k^2/n)\right) \\ &\le \left(1 + o(1)\right) \frac{1}{\sqrt{2\pi}\, c k^{5/2}}\, n (c\mathrm{e}^{1-c})^k\end{aligned}$$

This scales as $n\lambda^k$ where

$$\lambda = c\mathrm{e}^{1-c} ,$$

and $\lambda < 1$ since $\ln c < c - 1$. The expected number of components of size $A \log n$ or greater is then

$$\sum_{k=A\ln n}^{n} \mathbb{E}[T_k] = O(n\lambda^{A\log n}) = O(n^{1+A\log\lambda}) ,$$

which is $o(1)$ whenever $A > 1/(-\log\lambda)$.

14.5 The equation follows from the same calculation as in the previous problem. Note that we only get the correct exponent 5/2 if we use the term $\sqrt{2\pi n}$ in Stirling's approximation.

Ignoring constants, the expected number of components size greater than $s = n^\beta$ is then

$$n\int_s^\infty \frac{\mathrm{d}k}{k^{5/2}} \sim \frac{n}{s^{3/2}} = n^{1-(3/2)\beta} ,$$

and this is $o(1)$ whenever $\beta > 2/3$.

14.6 The probability that v's branching process dies out is $1-\gamma$, the probability that v is not in the giant component. If v has j children, the probability that this happens to all of them is $(1-\gamma)^j$. The conditional probability that v has j children is then Poisson-distributed with mean $c(1-\gamma)$,

$$\frac{1}{1-\gamma}\frac{\mathrm{e}^{-c}\,c^j\,(1-\gamma)^j}{j!}=\frac{\mathrm{e}^{-c(1-\gamma)}\,(c(1-\gamma))^j}{j!}\,,$$

where we used $1-\gamma=\mathrm{e}^{-c\gamma}$. Thus we set $d=c(1-\gamma)$, and check that

$$d\,\mathrm{e}^{-d}=c(1-\gamma)\mathrm{e}^{-c(1-\gamma)}=c\,\mathrm{e}^{-c}\,.$$

For another way to see this, the calculation in Problem 14.3 of the expected number of trees that span components of size k, which coincides with the expected number of components when k is small, holds for any c. We have $\mathbb{E}[T_k]\sim\lambda^k$ where $\lambda(c)=c\mathrm{e}^{1-c}<1$; it's just that if $c>1$ the expected total number of vertices in these small components sums to less than n, leaving room for the giant component. For any $\lambda<1$ there are two values of c such that $\lambda(c)=\lambda$, and these are the dual values $c>1>d$.

14.7 Using Cayley's formula, the expected number of spanning trees is

$$\mathbb{E}[T]=n^{n-2}\,p^{n-1}=\frac{c^{n-1}}{n}\,.$$

Analogous to Problem 14.2, the expected number of Hamiltonian cycles is

$$\mathbb{E}[H]=\frac{n!}{2n}\,p^n\geq\frac{n^n\,\mathrm{e}^{-n}}{2n}\,p^n=\frac{1}{2n}\left(\frac{c}{\mathrm{e}}\right)^n\,.$$

Thus $\mathbb{E}[T]$ and $\mathbb{E}[H]$ are exponentially large when $c>1$ or $c>\mathrm{e}$ respectively.

We can use the second moment method to show that $G(n,p=c/n)$ is disconnected with high probability, simply because an isolated vertex probably exists. If X is the number of isolated vertices, we have

$$\begin{aligned}\mathbb{E}[X]&=n(1-p)^{n-1}\\ \mathbb{E}[X^2]&=n(1-p)^{n-1}+n(n-1)(1-p)^{2n-1}\\ &=\big(1+O(1/n)\big)\,\mathbb{E}[X]^2\,,\end{aligned}$$

since a distinct pair of vertices are both isolated if the $2n-2$ edges between them and the other vertices, and the edge between them, are absent. Thus

$$\Pr[X>0]\geq\frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}=1-O(1/n)\,.$$

In fact, one can show that the probability that $G(n,p=c/n)$ is connected is exponentially small.

14.8 If $p=(1+\varepsilon)\ln n/n$, the expected number of subsets of size k disconnected from the rest of the graph is

$$\mathbb{E}[X_k]=\binom{n}{k}(1-p)^{k(n-k)}\leq\binom{n}{k}\mathrm{e}^{-pk(n-k)}=\binom{n}{k}n^{-(1+\varepsilon)(1-k/n)k}\,.$$

When k is small enough, we can use the extremely naive bound $\binom{n}{k} \le n^k$, and get

$$\mathbb{E}[X_k] = n^{(1-(1+\varepsilon)(1-k/n))k}$$

If $k < \delta n$ and $(1+\varepsilon)(1-\delta) > 1$, then $\mathbb{E}[X_k] \le n^{-\alpha k}$ for some $\alpha > 0$. Summing over all $k \ge 1$, the total contribution of these terms is $n^{-\alpha} = o(1)$.

When $\delta n \le k \le n/2$, we use the equally naive bound $\binom{n}{k} \le 2^n$, and get

$$\mathbb{E}[X_k] \le 2^n \, n^{-(1+\varepsilon)(1-\delta)\delta n} = n^{-\Omega(n)} .$$

Summing over all such k gives a total contribution which is still $n^{-\Omega(n)}$.

For the converse, if $p = (1-\varepsilon)\ln n/n$ then the expected number of isolated vertices is

$$n(1-p)^{n-1} \approx n\mathrm{e}^{-pn} = n\mathrm{e}^{-(1-\varepsilon)\ln n} = n^{\varepsilon} .$$

The events that a given pair of vertices are both isolated are nearly independent, so as in Problem 14.7 we can use the second moment method to show that an isolated vertex exists with probability $1 - O(p)$ whenever $\varepsilon > 0$.

14.9 By the union bound, the expected number of such subgraphs is at most

$$\binom{n}{s}\binom{\binom{s}{2}}{3s/2} p^{3s/2} \le \left(\frac{\mathrm{e}n}{s}\right)^{s} \left(\frac{\mathrm{e}s}{3}\right)^{3s/2} \left(\frac{c}{n}\right)^{3s/2} \le \left(\frac{\mathrm{e}^5 c^3 s}{n}\right)^{s/2} = \left(\frac{as}{n}\right)^{s/2}$$

where $a = \mathrm{e}^5 c^3$. If sum this from $s = 4$ to εn where $\varepsilon = 1/(2a)$, say, so that this series decreases geometrically, we get $O(n^{-2})$. Thus with high probability no such graph with $s \le \varepsilon n$ exists.

14.14 For the upper bound, if $n = R(k,\ell-1) + R(k-1,\ell)$ then any vertex v either has at least $R(k,\ell-1)$ green edges or at least $R(k-1,\ell)$ red edges. If it has $R(k,\ell-1)$ green edges, then its neighbors either include a red K_k or a green $K_{\ell-1}$, and in the second case v and its neighbors have a green K_ℓ. Similarly, if v has $R(k-1,\ell)$ red edges then either its neighbors have a green K_ℓ, or it and its neighbors have a red K_k.

This shows that $R(k,\ell) \le R(k,\ell-1) + R(k-1,\ell)$. Given the base case $R(1,\ell) = R(k,1) = 1$, the bound $R(k,\ell) \le \binom{k+\ell-2}{k-1}$ then follows by induction.

For the lower bound, if each edge of K_n is colored red or green with probability $1/2$, then the expected number of monochromatic cliques K_k is

$$\mathbb{E}[X] = 2\binom{n}{k} 2^{-\binom{k}{2}} \le 2\left(\frac{\mathrm{e}n}{k}\frac{\sqrt{2}}{2^{k/2}}\right)^k ,$$

where we used the inequality $\binom{n}{k} \le (\mathrm{e}n/k)^k$. Thus $\mathbb{E}[X]$ is exponentially small in k whenever $n < 2^{k/2}\,k/(\sqrt{2}\mathrm{e})$. Since a random coloring of K_n's edges has, with high probability, no monochromatic K_k in this case, $R(k,k)$ must be at least $2^{k/2}\,k/(\sqrt{2}\mathrm{e})$.

14.18 The differential equations are

$$\begin{aligned}\frac{\mathrm{d}s_k}{\mathrm{d}t} &= \frac{-ks_k}{1-t}\\ \frac{\mathrm{d}s_\ell}{\mathrm{d}t} &= \frac{1}{1-t}\left(\frac{\ell+1}{2}s_{\ell+1} - \ell s_\ell\right) \quad 2 \le \ell < k .\end{aligned}$$

By induction on ℓ, one can show that with the initial conditions $s_k(0) = \alpha$ and $s_\ell(0) = 0$ for $2 \le \ell < k$ the solution is

$$s_\ell(t) = \frac{\alpha}{2^{k-\ell}} \binom{k}{\ell} (1-t)^\ell \, t^{k-\ell} .$$

The ratio of the branching process is then

$$\lambda(t) = \frac{s_2(t)}{1-t} = \frac{\alpha k(k-1)}{2^{k-1}} (1-t) \, t^{k-2} .$$

This is maximized at $t = (k-2)/(k-1)$, and setting $\lambda = 1$ there gives

$$\alpha = \frac{1}{2} \left(\frac{k-1}{k-2} \right)^{k-2} \frac{2^k}{k} .$$

In the limit $k \to \infty$, the constant in front of $2^k/k$ becomes

$$\frac{1}{2} \left(\frac{k-1}{k-2} \right)^{k-2} = \frac{1}{2} \left(\frac{1-1/k}{1-2/k} \right)^{k-2} \approx \frac{1}{2} \left(\mathrm{e}^{1/k} \right)^{k-2} \approx \frac{\mathrm{e}}{2} ,$$

completing the proof.

14.19 Since the expected number of ℓ-clauses hit on each step is $\ell s_\ell/(1-t)$, and since with probability $1/2$ each one generates $\ell - 1$ unit clauses, we have

$$\lambda = \frac{1}{2} \sum_{\ell=2}^{k} \frac{\ell(\ell-1) s_\ell}{1-t} = \alpha \binom{k}{2} (1-t)(1-t/2)^{k-2} .$$

This is maximized at $t = 0$, in which case $\lambda < 1$ if and only if $\alpha \le 1/\binom{k}{2}$.

By the same token, if $\alpha > 1/\binom{k}{2}$ then satisfying any literal sets off a supercritical branching process. With constant probability this branching process forces us to set $\Theta(n)$ variables, almost certainly creating a contradiction. For a constant fraction of the variables, this happens whether we set it `true` or `false`, and the formula is unsatisfiable.

14.30 Given an assignment σ, the probability that it satisfies a random clause is the probability that it agrees with that clause on exactly $k/2$ variables. There are $\binom{k}{k/2}$ ways to do this and each one occurs with probability 2^{-k}, so the expected number of satisfying assignments is

$$\mathbb{E}[Z] = 2^n \left(2^{-k} \binom{k}{k/2} \right)^m = \left[2 \left(2^{-k} \binom{k}{k/2} \right)^\alpha \right]^n .$$

This is exponentially small whenever

$$\alpha > \alpha^* = \frac{\ln 2}{-\ln \left(2^{-k} \binom{k}{k/2} \right)} ,$$

and for $k=4$ this gives $\alpha_c \le 0.706695\ldots$ When k is large, applying Stirling's approximation to $\binom{k}{k/2}$ gives

$$\alpha^* \approx \frac{\ln 2}{\ln\sqrt{\pi k/2}} \approx \frac{2\ln 2}{\ln k}\,.$$

Given two assignments σ, τ with overlap ζ, the probability that they agree on j of the k variables in the clause is $\binom{k}{j}\zeta^j(1-\zeta)^{k-j}$. In that event, in order to satisfy a random clause c, they need to agree with c on $j/2$ of the variables they agree with each other on, as well as $(k-j)/2$ of the variables they disagree on. The number of ways for this to happen is $\binom{j}{j/2}\binom{k-j}{(k-j)/2}$, and each one occurs with probability 2^{-k}.

Thus the probability that σ and τ both satisfy a random clause is

$$\begin{aligned} q(\zeta) &= 2^{-k}\sum_{j=0,2,4,\ldots}^{k} \zeta^j(1-\zeta)^{k-j}\binom{k}{j}\binom{j}{j/2}\binom{k-j}{(k-j)/2} \\ &= 2^{-k}k!\sum_{j=0,2,4,\ldots}^{k}\frac{\zeta^j(1-\zeta)^{k-j}}{(j/2)!^2((k-j)/2)!^2}\,. \end{aligned}$$

For $k=4$, this gives the stated $q(\zeta)$. If we plot $\phi(\zeta)=h(\zeta)+\alpha\ln q(\zeta)$, we find that for $\alpha=0.601663$ the side peaks at $\zeta=0.135$ and $1-0.135$ are still below $\phi(1/2)$.

14.32 There are $\binom{n}{\alpha n}$ sets of αn vertices, and the probability that a given one is independent is $(1-p)^{\binom{\alpha n}{2}}$. Using $\binom{n}{k} \le (\mathrm{e}n/k)^k$ and ignoring constant factors, the expected number of such sets is then

$$\mathbb{E}[Z] = \binom{n}{\alpha n}(1-p)^{\binom{\alpha n}{2}} \le (\mathrm{e}/\alpha)^{\alpha n}\,\mathrm{e}^{-\alpha^2 cn/2} = \left[\mathrm{e}^{1-\ln\alpha-\alpha c/2}\right]^{\alpha n},$$

and this is exponentially small if $\ln\alpha + \alpha c/2 > 1$. In particular, if $\alpha = 2\ln c/c$ then

$$\ln\alpha + \alpha c/2 = \ln(2\ln c)\,,$$

and this exceeds 1 whenever $c > \mathrm{e}^{\mathrm{e}/2}$.

Using the better approximation $\binom{n}{\alpha n} \approx \mathrm{e}^{nh(\alpha)}$ where $h(\alpha) = -\alpha\ln\alpha - (1-\alpha)\ln(1-\alpha)$ and ignoring polynomial terms gives a sharper bound for finite c, namely the root α of $h(\alpha) - \alpha^2 c/2$. The approximation $\binom{n}{\alpha n} \le (\mathrm{e}/\alpha)^{\alpha n}$ corresponds to the inequality $h(\alpha) \le \alpha(1-\ln\alpha)$. However, for large c this doesn't change the factor 2 in front of $\ln c/c$.

14.33 If there are U remaining vertices, the expected number of them connected to the chosen vertex is pU. Since the chosen vertex is also removed, the expected change in U is

$$\mathbb{E}[\Delta U] = -1 - pU\,,$$

and rescaling to $U = u/n$ gives

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -1 - cu\,.$$

Solving this with the initial condition $u(0)=1$ gives

$$u(t)=\frac{(c+1)\mathrm{e}^{-ct}-1}{c}.$$

This algorithm finds an independent set of size T where $U(T)=0$. This is αn where $u(\alpha)=0$, which gives

$$\alpha=\frac{\ln(c+1)}{c}.$$

14.38 Substituting $y=B\theta$ and $\varepsilon=\frac{1}{B}$ in g we get

$$g(y)=\frac{\varepsilon}{2}\left(1-\cos y+\sin y\,\cot\frac{\varepsilon}{2}y\right).$$

A Taylor series expansion in ε yields

$$g(y)=\frac{\sin y}{y}+\frac{1-\cos y}{2}\varepsilon-\frac{y\sin y}{12}\varepsilon^2-\frac{y^3\sin y}{720}\varepsilon^4+\ldots=\frac{\sin y}{y}+f(y)\varepsilon).$$

where we have collected all terms that depend on ε in $f(y)\varepsilon$. Now consider

$$g^n(y)=\left(\frac{\sin y}{y}+f(y)\varepsilon\right)^n=\left(\frac{\sin y}{y}\right)^n+O(\varepsilon).$$

The integral (14.52) then reads

$$\int g^n(\theta)\mathrm{d}\theta=\varepsilon\int g^n(y)\mathrm{d}y=\varepsilon\int\left(\frac{\sin y}{y}\right)^n\mathrm{d}y+O(\varepsilon^2).$$

The function $\sin(y)/y$ has a unique maximum at $y=0$,

$$\frac{\sin y}{y}=1-\frac{1}{6}y^2+O(y^4),$$

and Laplace's method gives

$$\int\left(\frac{\sin y}{y}\right)^n\mathrm{d}y\approx\sqrt{\frac{6\pi}{n}}.$$

This confirms our result (14.55).

14.39 Let D denote the difference of a partition A on instance $\{a_j\}$,

$$D(A,\{a_j\})=\sum_{j\in A}a_j-\sum_{j\notin A}a_j.$$

We will construct a one-to-one mapping between instances that have a partition with $D=0$ and those that have a partition with $D=1$.

Let's assume that $D(A,\{a_j\})=0$. Set

$$a_1' = \begin{cases} a_1+1 & \text{if } 1\in A, \\ a_1-1 & \text{if } 1\notin A, \end{cases}$$

and $a_j' = a_j$ for $j>1$. Then obviously $D(A,\{a_j'\})=1$. For the inverse mapping assume that $D(A,\{a_j\})=0$. Set

$$a_1' = \begin{cases} a_1-1 & \text{if } 1\in A, \\ a_1+1 & \text{if } 1\notin A, \end{cases}$$

and $a_j' = a_j$ for $j>1$. Then $D(A,\{a_j'\})=1$. Since this is a one-to-one mapping, the number of instances with $D=1$ equals the number of instances with $d=0$. Note that the mapping may fail if we want to decrease a_1 by one but $a_1=0$, or if we want to increase a_1 but $a_1=B-1$. Hence our mapping fails with probability $1-1/B$, which explains the factor in $\Pr[X_1>0]=(1-O(1/B))\Pr[X_0>0]$.

14.41 We have

$$\mathbb{E}[Z]=\mathbb{E}[Z(1-Y)]+\mathbb{E}[ZY]\le\theta\,\mathbb{E}[Z]+\mathbb{E}[ZY]\le\theta\,\mathbb{E}[Z]+\sqrt{\mathbb{E}[Z^2]\,\mathbb{E}[Y^2]}\,.$$

Rearranging and using $\mathbb{E}[Y^2]=\mathbb{E}[Y]=\Pr\left[Z>\theta\,\mathbb{E}[Z]\right]$ completes the proof.

14.42 Select an arbitrary constraint vertex a. Since the factor graph is a tree, we have

$$E^* = \min_{X_{\partial a}}\left(E_a(X_{\partial a})+\sum_{i\in\partial a}E_{i\to a}(x_i)\right),$$

where $E_{i\to a}(x_i)$ is the minimum of the subproblem that is encoded by the tree rooted in variable i where i has the value x_i. This message can be calculated from messages farther up the tree,

$$E_{i\to a}(x_i)=\sum_{b\in\partial\backslash a}E_{b\to i}(x_i)\,.$$

Here $E_{b\to i}(x_i)$ is the minimum value of the subtree rooted in b such that i has the value x_i, and these messages can be calculated according to

$$E_{a\to i}(x_i)=\min_{X_{\partial a\backslash i}}\left(E_a(X_{\partial a})+\sum_{i\in\partial a\backslash i}E_{i\to a}(x_i)\right).$$

You see why we call these "min-sum" equations. Once we've calculated the bidirectional messages for each edge, we can compute the optimal solution X^* from

$$x_i^* = \operatorname{argmin}_{x_i}\sum_{b\in\partial i}E_{b\to i}(x_i)\,.$$

14.43 According to (14.77a), $\widehat{\eta} = O(\varepsilon)$ if and only if all factors $\eta_1, \ldots, \eta_{k-1} = \Omega(1-\varepsilon)$. Hence

$$y = x^{k-1}.$$

Let p' and q' denote the number of messages $\widehat{\eta} = O(\varepsilon)$ on the rhs of (14.77b). Then we can write (14.77b) as

$$\eta = \frac{O\left(\varepsilon^{p'}\right)\Omega\left((1-\varepsilon)^{q'}\right)\Theta(1)}{O\left(\varepsilon^{p'}\right)\Omega\left((1-\varepsilon)^{q'}\right)\Theta(1) + \Omega\left((1-\varepsilon)^{p'}\right)O\left(\varepsilon^{q'}\right)\Theta(1)}.$$

If $p' < q'$, we cancel a factor $\varepsilon^{p'}$ and obtain $\eta = \Omega(1-\varepsilon)$. If $p' > q'$, we cancel a factor $\varepsilon^{q'}$ and obtain $\eta = O(\varepsilon)$. Finally, for $p' = q'$ we cancel a factor $\varepsilon^{p'}$ and obtain $\eta = \Theta(1)$. Since the events $p' > q'$ and $p' < q'$ have the same probability, we have

$$y = \Pr[\eta = O(\varepsilon)] = \Pr[\eta = \Omega(1-\varepsilon)]$$

and therefore

$$1 - 2y = \Pr[\eta = \Theta(1)].$$

The event $\eta = \Theta(1)$ is equivalent to the event $p' = q'$ and the events that p' out of p and q' out of q variables $\widehat{\eta}$ are $O(\eta)$. Setting $\lambda = k\alpha/2$, this probability is

$$\begin{aligned}
1 - 2y &= \sum_{p=0}^{\infty} \frac{\lambda^p}{p!} \mathrm{e}^{-\lambda} \sum_{q=0}^{\infty} \frac{\lambda^q}{q!} \mathrm{e}^{-\lambda} \sum_{p'=0}^{p} \sum_{q'=0}^{q} \delta_{p'q'} \binom{p}{p'} x^{p'}(1-x)^{p-p'} \binom{q}{q'} x^{q'}(1-x)^{q-q'} \\
&= \mathrm{e}^{-2\lambda} \sum_{p'=0}^{\infty} \sum_{q'=0}^{\infty} \sum_{p=p'}^{\infty} \sum_{q=q'}^{\infty} \frac{\lambda^p}{p!} \frac{\lambda^q}{q!} \delta_{p'q'} \binom{p}{p'} \binom{q}{q'} x^{p'+q'} (1-y)^{p+q-p'-q'} \\
&= \mathrm{e}^{-2\lambda} \sum_{t=0}^{\infty} \sum_{p=t}^{\infty} \sum_{q=t}^{\infty} \frac{\lambda^p \lambda^q}{(t!)^2 (p-t)!(q-t)!} x^{2t} (1-x)^{p+q-2t} \\
&= \mathrm{e}^{-2\lambda} \sum_{t=0}^{\infty} \sum_{s=0}^{\infty} \sum_{u=o}^{\infty} \frac{\lambda^{s+t} \lambda^{u+t}}{(t!)^2 s! u!} x^{2t} (1-x)^{s+u} \\
&= \mathrm{e}^{-2\lambda} \sum_{t=0}^{\infty} \frac{(\lambda x)^{2t}}{(t!)^2} \left(\sum_{s=0}^{\infty} \frac{\lambda^s}{s!} (1-x)^s \right)^2 \\
&= \mathrm{e}^{-2\lambda} I_0(2\lambda x) \mathrm{e}^{2\lambda(1-x)} \\
&= \mathrm{e}^{-2\lambda x} I_0(2\lambda x)
\end{aligned}$$

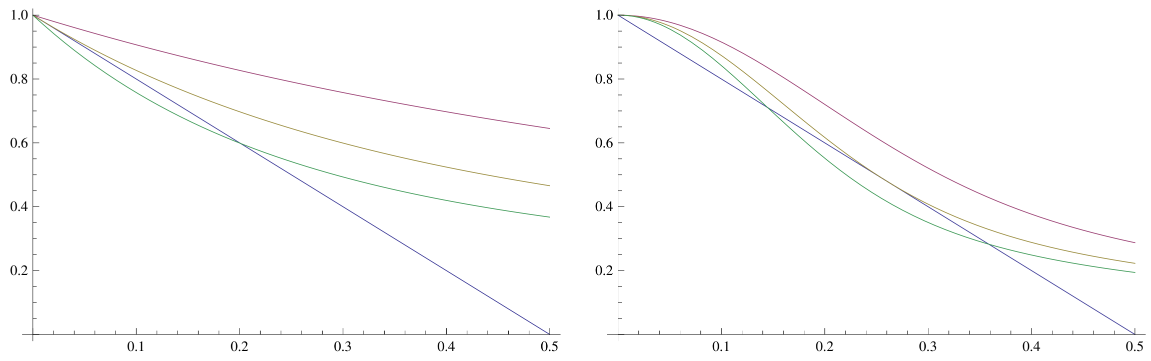
We define

$$F_{k,\alpha}(y) = \mathrm{e}^{-k\alpha y^{k-1}} I_0(k\alpha y^{k-1}).$$

We look for the solutions of $1-2y = F_{k,\alpha}(y)$. Plots of $1-2y$ and $F_{k,\alpha}(y)$ (see Figure) tell us, that for $\alpha < \alpha_{\mathrm{BP}}$, there is only the solution $y = 0$, and for $\alpha > \alpha_{\mathrm{BP}}$ nonzero solutions appear.

Let $y^\star(k)$ denote the new solution that emerges at $\alpha = \alpha_{\mathrm{BP}}$. We get both values $y^\star$ and α_{BP} as the solution of

$$\begin{aligned}
1 - 2y^\star &= F_{k\alpha_{\mathrm{BP}}}(y^\star) \\
-2 &= F'_{k\alpha_{\mathrm{BP}}}(y^\star).
\end{aligned}$$



Plot of $1-2y$ and $F_{k,\alpha}(y)$ for $k=2$ and $\alpha=0.5, 1.0, 1.5$ (left) and $k=3$ and $\alpha=3.0, 4.667, 6.0$ (right).

Solving these equations numerically yields

$$\begin{aligned} \alpha_{\mathrm{BP}}(3) &= \ \ 4.6672805\ldots & y^\star(3) &= 0.26021390\ldots \\ \alpha_{\mathrm{BP}}(4) &= 11.8322991\ldots & y^\star(4) &= 0.32296033\ldots \end{aligned}$$

For $k=2$, the nonzero solution evolves continuously from zero, i.e., we have $y^\star(2)=0$. But $F'_{2,\alpha}(0)=-2\alpha$, hence

$$\alpha_{\mathrm{BP}}(2)=1\,.$$

14.44 To quantify the clustering transition we note that a random vertex of the cube on average belongs to

$$2^{(1-\alpha)n}\left(1-\frac{p}{2}\right)^n = 2^{(\log_2(2-p)-\alpha)n}$$

clusters. Therefore, if

$$\alpha < \alpha_{\mathrm{clust}} := \log_2(2-p)\,,$$

then with high probability *every* vertex is an element of the solutions space $\mathscr{S}$. No clustering, and the total entropy s_{tot} equals 1.

If $\alpha > \alpha_{\mathrm{clust}}$, the probability that a random vertex is a solution, is exponentially small. And the probability that a given solution belongs to more than one subcube is also exponentially small. Hence above α_{clust}, the solution space is composed of non-overlapping subcubes which we call clusters.

The internal entropy density $s = n^{-1}\log_2|A|$ of a cluster equals the fraction of its free variables. The probability $P(s)$ that a cluster has internal entropy s is given by the binomial distribution

$$P(s) = \binom{n}{sn}(1-p)^{sn}p^{(1-s)n}\,.$$

The number $N(s)$ of clusters with entropy s follows a binomial distribution with parameter $P(s)$ and $2^{(1-\alpha)n}$ terms. Mean and variance of $N(s)$ are

$$\mathbb{E}[N(s)] = 2^{(1-\alpha)n}P(s) \quad\text{and}\quad \mathrm{Var}[N(s)] = 2^{(1-\alpha)n}P(s)\big(1-P(s)\big)\,.$$

We write the mean value as $\mathbb{E}[N(s)] = 2^{n\Sigma(s)}$ with $\Sigma(s) = 1-\alpha+n^{-1}\log_2 P(s)$ or

$$\Sigma(s) = 1-\alpha - s\log_2\frac{s}{1-p} - (1-s)\log_2\frac{1-s}{p} + O(\frac{\log n}{n}).$$

This is a convex function that has two roots $s_{\min} \le s_{\max}$ for $\alpha < \alpha_c$, see Figure 14.28.

The first moment bound

$$\Pr[N(s)\ge 1] \le \mathbb{E}[N(s)] = 2^{n\Sigma(s)}$$

implies that for large n there are no clusters whith $\Sigma(s)<0$. For $\Sigma(s)>0$, on the other hand, Chebyshev's inequality (A.15) tells us that the number $N(s)$ of clusters with internal entropy s is concentrated at its mean value,

$$\Pr\left[\left|\frac{N(s)}{\mathbb{E}[N(s)]}-1\right| > \varepsilon\right] \le \frac{\mathrm{Var}[N(s)]}{\mathbb{E}[N(s)]^2\varepsilon^2} \le \frac{1}{2^{(1-\alpha)n}P(s)} = 2^{-n\Sigma(s)}$$

for all $\varepsilon>0$. Hence in the limit $n\to\infty$, the complexity $\Sigma(s)$ gives the logarithm of the number of clusters with internal entropy s,

$$\lim_{n\to\infty}\frac{1}{n}\log_2(\text{\# of clusters of size } s) = \begin{cases}\Sigma(s) & s_{\min}\le s\le s_{\max}\\ 0 & \text{otherwise}\end{cases}$$

The maximum of $\Sigma(s)+s$ is located at $s^\star = 2(1-p)/(2-p)$, the solution of $\partial_s\Sigma(s^\star) = -1$. The clustered phase is given by the condition $\Sigma(s^\star)\ge 0$, which is equivalent to

$$\alpha \le \alpha_{\text{cond}} := \frac{p}{p-2} + \log_2(2-p).$$

14.45 We can write

$$\zeta = \frac{1}{n}\sum_i\sum_{x_i}\delta(x_i,x'_i),$$

where $\delta(x_i,x'_i)=1$ if $x_i = x'_i$ and 0 if $x_i\ne x'_i$. We have $\mathbb{E}[\delta(x_i,x'_i)] = \sum_{x_i}\mu^2(x_i)$ and $\mathbb{E}[\delta(x_i,x'_i)\delta(x_j,x'_j)] = \sum_{x_i,x_j}\mu^2(x_i,x_j)$, so the expectation and second moment of ζ are then

$$\mathbb{E}[\zeta] = \frac{1}{n}\sum_i\sum_{x_i}\mu^2(x_i) \quad\text{and}\quad \mathbb{E}[\zeta^2] = \frac{1}{n^2}\sum_{i,j}\sum_{x_i,x_j}\mu^2(x_i,x_j),$$

and (14.100) follows from $\mathrm{Var}\,\zeta = \mathbb{E}[\zeta^2]-\mathbb{E}[\zeta]^2$.

Each term in (14.100) is large only if x_i and x_j are strongly correlated, i.e., if $\left|\mu(x_i,x_j)-\mu(x_i)\mu(x_j)\right| = \Theta(1)$. In the clustered phase, the probability that $\mathbf{x}$ and $\mathbf{x}'$ are in the same cluster is exponentially small, so $\zeta = \zeta_2 + O(1/\sqrt{n})$ with high probability and $\mathrm{Var}\,\zeta = O(1/n)$. It follows that only a fraction $O(1/n)$ of the pairwise correlations are large, so on average a given variable x_i is correlated with $O(1)$ others.

In the condensed phase, $\mathbf{x}$ and $\mathbf{x}'$ are in the same cluster with constant probability, so $\mathrm{Var}\,\zeta = \Theta(1)$. Then $\Theta(n^2)$ of the pairwise correlations are large, and a typical variable is correlated with $\Theta(n)$ others. If the factor graph were r-regular then there are at most r^ℓ variables ℓ steps away, and setting $r^\ell = \Theta(n)$ gives $\ell = \Omega(\log n)$.

Chapter 15

Quantum Computation

15.3 One way to do this is to construct a square root of Z, such as

$$\begin{pmatrix} 1 & 0 \\ 0 & \imath \end{pmatrix}$$

and then transform it with H. This gives

$$V = \frac{1}{2}\begin{pmatrix} 1+\imath & 1-\imath \\ 1-\imath & 1+\imath \end{pmatrix}$$

Of course, this is not unique. How many square roots does X have?

15.4 We can explicitly multiply the matrices corresponding to each gate:

$$\begin{pmatrix} \mathbb{1} & & & \\ & V & & \\ & & \mathbb{1} & \\ & & & V \end{pmatrix} \cdot \begin{pmatrix} \mathbb{1} & & & \\ & \mathbb{1} & & \\ & & & \mathbb{1} \\ & & \mathbb{1} & \end{pmatrix} \cdot \begin{pmatrix} \mathbb{1} & & & \\ & V^\dagger & & \\ & & \mathbb{1} & \\ & & & V^\dagger \end{pmatrix}$$
$$\cdot \begin{pmatrix} \mathbb{1} & & & \\ & \mathbb{1} & & \\ & & & \mathbb{1} \\ & & \mathbb{1} & \end{pmatrix} \cdot \begin{pmatrix} \mathbb{1} & & & \\ & \mathbb{1} & & \\ & & V & \\ & & & V \end{pmatrix} = \begin{pmatrix} \mathbb{1} & & & \\ & \mathbb{1} & & \\ & & \mathbb{1} & \\ & & & V \end{pmatrix}.$$

A simpler way is to note that each control qubit controls one of the outer two controlled-V gates, and that the controlled-$V^\dagger$ gate is applied if the two control qubits are different. Note also that the second control qubit is always returned to its original value, since we apply a controlled-NOT to it twice.

15.5 We just need to compare inner products before and after U, which are equal if U is unitary. Before, we have

$$\langle v \otimes b \,|\, w \otimes b \rangle = \langle v \,|\, w \rangle \langle b \,|\, b \rangle = \langle v \,|\, w \rangle$$

and after we have

$$\langle v \otimes v \,|\, w \otimes w \rangle = \langle v \,|\, w \rangle^2 .$$

Since $\langle v \,|\, w \rangle = \langle v \,|\, w \rangle^2$, it must be 1 or 0.

15.7 Since $A_1 B_2 = A \otimes B$, we have

$$\begin{aligned}
\mathbb{E}[A_1 B_2] &= \langle \psi | A \otimes B | \psi \rangle \\
&= \frac{1}{2}\Big(\langle 0|A|0\rangle\langle 0|B|0\rangle + \langle 0|A|1\rangle\langle 0|B|1\rangle + \langle 1|A|0\rangle\langle 1|B|0\rangle + \langle 1|A|1\rangle\langle 1|B|1\rangle \Big) \\
&= \frac{1}{2}\Big(\langle 0|A|0\rangle\langle 0|B^T|0\rangle + \langle 0|A|1\rangle\langle 1|B^T|0\rangle + \langle 1|A|0\rangle\langle 0|B^T|1\rangle + \langle 1|A|1\rangle\langle 1|B^T|1\rangle \Big) \\
&= \frac{1}{2}\Big(\langle 0|AB^T|0\rangle + \langle 1|AB^T|1\rangle \Big) \\
&= \frac{1}{2}\,\mathrm{tr}(AB^T)
\end{aligned}$$

15.12 The two factors of -1 in $XZ = -ZX$ cancel when we commute $X \otimes X$ and $Z \otimes Z$. To be more explicit,

$$(X \otimes X)(Z \otimes Z) = XZ \otimes XZ = (-ZX) \otimes (-ZX) = ZX \otimes ZX = (Z \otimes Z)(X \otimes X) .$$

Their eigenvectors are exactly the Bell basis states $|\psi_{a,b}\rangle$ defined in Section 15.3.4. Specifically,

$$(X \otimes X)|\psi_{a,b}\rangle = (-1)^b |\psi_{a,b}\rangle \;\text{ and } (Z \otimes Z)|\psi_{a,b}\rangle = (-1)^a |\psi_{a,b}\rangle .$$

15.22 We prove (15.70) by induction. The first convergents are

$$\begin{aligned}
\frac{p_0}{q_0} &= c_0 = \frac{c_0}{1} \\
\frac{p_1}{q_1} &= c_0 + \frac{1}{c_1} = \frac{c_1 c_0 + 1}{c_1} \\
\frac{p_2}{q_2} &= c_0 + \frac{1}{c_1 + \frac{1}{c_2}} = \frac{c_2 p_1 + p_0}{c_2 q_1 + q_0} ,
\end{aligned}$$

which proves the base case $n = 2$. Now let us assume that

$$\frac{p_n}{q_n} = \frac{c_n p_{n-1} + p_{n-2}}{c_n p_{n-1} + p_{n-2}} .$$

Obviously, we can get the next convergent by replacing c_n by $c_n + 1/c_{n+1}$:

$$\begin{aligned}
\frac{p_{n+1}}{q_{n+1}} &= \frac{\left(c_n + \frac{1}{c_{n+1}}\right) p_{n-1} + p_{n-2}}{\left(c_n + \frac{1}{c_{n+1}}\right) p_{n-1} + p_{n-2}} \\
&= \frac{\overbrace{c_n p_{n-1} + p_{n-2}}^{p_n} + \frac{1}{c_{n+1}} p_{n-1}}{\underbrace{c_n q_{n-1} + q_{n-2}}_{q_n} + \frac{1}{c_{n+1}} q_{n-1}} \\
&= \frac{c_{n+1} p_n + p_{n-1}}{c_{n+1} q_n + q_{n-1}} .
\end{aligned}$$

This completes the proof of (15.70).

The strict monotinicity of (q_n) follows directly from (15.70) and $c_k > 0$ for $k \geq 1$.

The matrix product (15.71) is also proven by induction. The inductive step is

$$\begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix} = \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} \begin{pmatrix} c_n & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} c_{n+1}p_n + p_{n-1} & p_n \\ c_{n+1}q_n + q_{n-1} & q_n \end{pmatrix}.$$

Taking the determinants of all matrices in (15.71) provides us with

$$\det C_n = \begin{vmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{vmatrix} = p_n q_{n-1} - q_n p_{n-1} = (-1)^{n+1}, \tag{a}$$

i.e., the matrix C_n is unimodular. We also have

$$\frac{p_n}{q_n} = \frac{p_{n-1}}{q_{n-1}} + \frac{(-1)^{n-1}}{q_n q_{n-1}}$$

or

$$\frac{p_n}{q_n} = c_0 + \sum_{k=1}^{n} \frac{(-1)^{k+1}}{q_k q_{k-1}}.$$

Since the series p_n/q_n converges to x, we have

$$x - \frac{p_n}{q_n} = (-1)^n \left(\frac{1}{q_n q_{n+1}} - \frac{1}{q_{n+1} q_{n+2}} + \cdots \right).$$

This shows that $a_n = q_n x - p_n$ is an alternating sequence and $|a_n|$ is monotonically decreasing.

15.23 We start by showing that a convergent p_n/q_n is a locally best approximation. Let $p'/q' \neq p_n/q_n$ and $q' \leq q_n$. Then unimodularity (see (a) in the solution of the previous problem) guarantees that there are integers a and b such that

$$\begin{aligned} p' &= a p_n + b p_{n-1} \\ q' &= a q_n + b q_{n-1} \end{aligned}.$$

Since $b = 0$ would imply $p'/q' = p_n/q_n$, we can assume that $b \neq 0$. Furthermore we know that $ab \leq 0$ because $q' \leq q_n$. Multiplying the second equation by x and subtracting both equations gives us

$$q'x - p' = a(q_n x - p_n) + b(q_{n-1}x - p_{n-1}).$$

Since $q_n x - p_n$ is an alternating sequence, the two summands on the right hand sight have the same sign. This allows us to write

$$\left|q'x - p'\right| = |a| \left|q_n x - p_n\right| + |b| \left|q_{n-1}x - p_{n-1}\right|,$$

and, because $b \neq 0$,

$$\left|q'x - p'\right| \geq \left|q_{n-1}x - p_{n-1}\right| > \left|q_n x - p_n\right|, \tag{a}$$

where the second inequality is the monotonicity of the sequence $|a_n|$ we proved in Problem 15.22. This proves that a convergent is a locally best approximation.

We prove the converse by contradiction. Assume that p/q is a locally best approximation to x that is different from all convergents of x. We will find a convergent that is a locally better approximation than p/q, in contradiction to our assumption. For $q=1$, $p_0/q_0 = \lfloor x \rfloor$ will do the job because

$$\left|x-p\right| > |x-\lfloor x \rfloor| \qquad \text{for } p \neq \lfloor x \rfloor .$$

Hence we can assume that $q>1$. Let q_n be the first element in the monotonically increasing sequence of the denominators of the convergents such that

$$q_{n-1} < q \leq q_n .$$

Since $p/q \neq p_n/q_n$, we can apply (a) to get

$$\left|qx-p\right| \geq \left|q_{n-1}x-p_{n-1}\right| .$$

Thus p_n/q_n is a better approximation than p/q, violating our assumption that p/q is a locally best approximation.

15.24 Take integers r, s such that $0<s\leq q$ and $p/q \neq r/s$.

$$\begin{aligned} 1 \leq \left|qr-ps\right| &= \left|s(qx-p)-q(sx-r)\right| \\ &\leq s\left|qx-p\right| + q\,|sx-r| \\ &< \frac{s}{2q} + q\,|sx-r| . \end{aligned}$$

Hence we have

$$q\,|sx-r| > 1 - \frac{s}{2q} \geq \frac{1}{2}$$

and

$$|sx-r| > \frac{1}{2q} > \left|qx-p\right| .$$

15.26 From (15.73) we get

$$\begin{aligned} \frac{\varphi(N)}{N} &= \prod_{\substack{p|N \\ p>\log_2 N}} \left(1-\frac{1}{p}\right) \prod_{\substack{p|N \\ p\leq\log_2 N}} \left(1-\frac{1}{p}\right) \\ &\geq \left(1-\frac{1}{\log_2 N}\right)^{\log_2 N} \prod_{\substack{p|N \\ p\leq\log_2 N}} \left(1-\frac{1}{p}\right) \end{aligned}$$

Since $1/4 \leq (1-1/x)^x \leq \mathrm{e}^{-1}$ for all $x\geq 2$, the first term can be replaced by a constant $C \geq 1/4$. The remaining product gets smaller if we extend it to all integers $\leq \log_2 N$,

$$\begin{aligned} \frac{\varphi(N)}{N} &\geq C \prod_{i=2}^{\log_2 N} (1-1/i) \\ &= C\,\frac{1}{2}\,\frac{2}{3}\,\frac{3}{4}\cdots\frac{\log_2 N - 1}{\log_2 N} \\ &= \frac{C}{\log_2 N} . \end{aligned}$$

Asymptotically, this gives $C \approx \mathrm{e}^{-1}$, but this can be improved. Let ρ denote the number of large primes. We have

$$(\log_2 N)^\rho < \prod_{\substack{p|N \\ p>\log_2 N}} p \le N .$$

Taking the logarithm provides us with

$$\rho < \frac{\log_2 N}{\log_2 \log_2 N} .$$

Hence we get a more precise bound by

$$\prod_{\substack{p|N \\ p>\log_2 N}} \left(1-\frac{1}{p}\right) > \left(1-\frac{1}{\log_2 N}\right)^\rho > \left(1-\frac{1}{\log_2 N}\right)^{\frac{\log_2 N}{\log_2 \log_2 N}} .$$

The last expression converges to 1, so asymptotically C approaches 1.

Extending the product from all primes $p \le \log N$ to all integers $i \le \log N$ yields a nice telescoping product, but the resulting bound is not very tight. In Problem 10.24 we argued that

$$\prod_{p\le x}\left(1-\frac{1}{p}\right) \sim \frac{1}{\ln x} ,$$

which gives the stronger bound.

15.27 As discussed in the text, d is the inverse of e in $\mathbb{Z}^*_{\varphi(N)}$. In any group G, the generalized Little Theorem tells us that $e^{|G|} = 1$ for any $e \in G$. Therefore, $e^{|G|-1} = e^{-1}$. Observing that $\left|\mathbb{Z}^*_{\varphi(N)}\right| = \varphi(\varphi(N))$ completes the proof.

15.31 We can write $|k,\ell\rangle$ as

$$|k,\ell\rangle = |k\rangle \otimes |\ell\rangle = \frac{1}{d}\sum_{x,y=0}^{d-1} \omega^{kx+\ell y} |x,y\rangle .$$

Applying U and changing variables from y to $z = y + x$ gives

$$\begin{aligned} U|k,\ell\rangle &= \frac{1}{d}\sum_{x,y=0}^{d-1} \omega^{kx+\ell y}\, U|x,y\rangle \\ &= \frac{1}{d}\sum_{x,y=0}^{d-1} \omega^{kx+\ell y} |x,y+x\rangle \\ &= \frac{1}{d}\sum_{x,y=0}^{d-1} \omega^{kx+\ell(z-x)} |x,z\rangle \\ &= \frac{1}{d}\sum_{x,y=0}^{d-1} \omega^{(k-\ell)x+\ell z} |x,z\rangle \\ &= |k-\ell,\ell\rangle . \end{aligned}$$

15.32 Suppose we know m^x, m^{xy}, and m^y. Let a be a primitive root, and let $m = a^t$. Then taking discrete logs gives tx, txy, and ty. Since we can find inverses in $\mathbb{Z}_{p-1}^*$ using Euclid's algorithm, we can find $tx \cdot ty \cdot (txy)^{-1} = t$, and once we know t we can obtain $a^t = m$.

15.38 We can check to see whether there is a collision in K with $|K|$ classical queries. If there isn't, then according to Exercise 15.23, we can find one of the $|K|$ elements which collide with an element of K with $O(\sqrt{N/|K|})$ queries. Summing these two, we wish to minimize

$$|K| + O(\sqrt{N/|K|}).$$

This is minimized when both terms, and therefore $|K|$ as well, are $O(N^{1/3})$.

15.39 If we measure the second system and observe $|k\rangle$, the state collapses to $|v_k\rangle \otimes |k\rangle$. If $P(k) = \sum_i |a_{ik}|^2$ is the probability of observing $|k\rangle$, then

$$|v_k\rangle = \frac{1}{\sqrt{P(k)}} \sum_i a_{ik} |i\rangle .$$

Then setting

$$\rho = \sum_k P(k) |v_k\rangle\langle v_k|$$

the $P(k)$s cancel, giving

$$\rho_{ij} = \sum_k a_{ik} a_{jk}^*$$

as stated. If $a_{ik} = a_i \langle k|\psi_i\rangle$, this becomes

$$\rho_{ij} = a_i a_j^* \sum_k \langle k|\psi_i\rangle\langle\psi_j|k\rangle = a_i a_j^* \langle\psi_j|\psi_j\rangle .$$

15.46 If each attempt succeeds with probability $p = \sin^2 2a$, then the average number of attempts we need to make is $1/p$. Thus the total expected number of steps is $b\sqrt{N}$ where

$$b = \frac{a}{\sin^2 2a} .$$

This is minimized when $a \approx 0.583$ is the root of $\sin 2a - 4a \cos 2a = 0$, at which point $b \approx 0.690$.

15.48 Since H is a projection operator, we have $H^j = H$ for all $j \geq 1$. Then using the Taylor series,

$$e^{\imath Ht} = \sum_{j=0}^{\infty} \frac{(iHt)^j}{j!} = \mathbb{1} + H \sum_{j=1} \frac{(it)^j}{j!} = \mathbb{1} + (e^{\imath t} - 1)H .$$

When $t = \pi$, this is $\mathbb{1} - 2H$, which is $-D$. More generally, if $H = |v\rangle\langle v|$, then $e^{\imath H\pi} = -R_v$.

15.50 Setting $a = x/t$, we can write (15.58) as

$$\Psi_t(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{e}^{\imath t \phi(k)} \, \mathrm{d}k$$

where

$$\phi(k) = ak - \cos k \, .$$

If $|a| < 1$, there are two fixed points where $\phi'(k_0) = 0$, namely $k_0 = -\sin^{-1} a$ and its supplement $k = \pi - k_0$. At these points we have

$$\left|\phi''(k_0)\right| = |\cos k_0| = \sqrt{1-a^2} \, .$$

According to Appendix A.6.2, we then have

$$\Psi_t(x) \sim \frac{1}{\sqrt{t\left|\phi''(k_0)\right|}}$$

and so $P(x) = |\Psi_t(x)|^2$ scales as $1/(t\left|\phi''\right|) = 1/(t\sqrt{1-a^2})$ as stated.

At $a = \pm 1$ we have $k_0 = \pm\pi/2$. Then k_0 becomes a third-order fixed point, since $\phi''(k_0) = 0$ but $\left|\phi'''(k_0)\right| = 1$, so according to Appendix A.6.2 we have $\Psi_t \sim t^{-1/3}$ and $P_t \sim t^{-2/3}$. The same scaling holds if $t - |x| = t^{1/3}$, since then $|a| = 1 - O(t^{-2/3})$, $\left|\phi''\right| \sim t^{-1/3}$, and $1/\sqrt{t\left|\phi''\right|} \sim t^{-1/3}$.

15.53 The probability of success is a linear function of the M_i,

$$\sum_{i=1}^{t} p_i \, \mathrm{tr}(M_i \rho_i) = \sum_{i=1} \langle p_i \rho_i, M_i \rangle$$

with the inner product of matrices defined as $\langle A, B\rangle = \mathrm{tr}(A^\dagger B)$. The constraint $\sum_i M_i = \mathbb{1}$ gives d^2 constraints on the entries of the M_i if the Hilbert space is d-dimensional. The only other constraint is $M_i \succeq 0$ for all i. so this is an instance of SEMIDEFINITE PROGRAMMING.

Appendix A

Mathematical Tools

A.2 Let $T \subseteq \{1, \dots, n\}$, and let P_T be the probability that every opera-goer in T gets their own manteau back. If $|T| = i$ then there are exactly $(n-i)!$ permutations with this property, so $P_T = (n-i)!/n!$. Since there are $\binom{n}{i}$ subsets of size i, the inclusion–exclusion principle (A.13) gives

$$P = \sum_{i=0}^{n} (-1)^i \binom{n}{i} \frac{(n-i)!}{n!} = \sum_{i=0}^{n} \frac{(-1)^i}{i!} \,.$$

A.7 There are $\binom{n}{3}$ possible triples. The probability that any one triple all have the same birthday is $1/y^2$, since this is the probability that the second and third have the same birthday as the first. Thus the expected number of such triples is $\binom{n}{3} \frac{1}{y^2} = \Theta(n^3/y^2)$. This is $\Theta(1)$ when $n = \Theta(y^{2/3})$.

A.9 The order of a given set of k numbers is equally likely to be any of the $k!$ permutations, so the probability that they are in increasing order is $1/k!$. The expected number of such sets is then

$$\binom{n}{k} \frac{1}{k!} \,.$$

Using $\binom{n}{k} \le \left(\frac{\mathrm{e}n}{k}\right)^k$ and $k! \ge k^k \,\mathrm{e}^{-k}$, this is at most

$$\left(\frac{\mathrm{e}n}{k}\right)^k \frac{1}{k^k \,\mathrm{e}^{-k}} = \left(\frac{\mathrm{e}^2 n}{k^2}\right)^k \,.$$

Thus for any constant $A > \mathrm{e}$, if $k \ge A\sqrt{n}$ the expected number of such sets is $o(1)$.

A.12 By linearity of expectation, we have

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i} \Pr[i] = 2^{-k} n \,.$$

If $k \ge (1+\varepsilon)\log_2 n$, this is $\mathbb{E}[X] = n^{-\varepsilon} = o(1)$.

To calculate the second moment, we have to take into account the fact that pairs of events i and j are positively correlated to the extent that their strings overlap. If $|i-j|<k$ (with $|i-j|$ defined in a way that takes the wraparound into account) and there are k black beads starting at i, then we only need $|i-j|$ additional black beads to have a string at j as well. Thus, assuming $k \le n/2$,

$$\Pr[j\,|\,i] = \begin{cases} 2^{|i-j|} & \text{if } |i-j|<k \\ 2^{-k} & \text{if } |i-j|\ge k \end{cases} .$$

Therefore, using (A.20) we have

$$\begin{aligned} \mathbb{E}[X^2] &= 2^{-k} n \sum_{j=1}^{n} \Pr[j\,|\,i] \\ &= 2^{-k} n \left(1 + 2\sum_{\ell=1}^{k-1} 2^{-\ell} + (n-2k+1)2^{-k} \right) \qquad \text{(a)} \\ &= 2^{-k} n \left(3 + (n-2k-3)2^{-k} \right) \end{aligned}$$

Here the term 1 in (a) corresponds to the case $j=i$, the sum over ℓ corresponds to events with varying amounts of overlap with i, and the remaining terms correspond to events which have no overlap and so are independent of i.

Finally, applying (A.18) gives

$$\Pr[X>0] \ge \frac{\left(2^{-k}n\right)^2}{2^{-k}n\left(3+(n-2k-3)2^{-k}\right)} > \frac{2^{-k}n}{3+2^{-k}n} ,$$

from which the statement follows.

A.28 We have $x^n\,\mathrm{e}^{-x} = \mathrm{e}^{n\phi(x)}$ where $\phi(x) = \ln x - x/n$. Since

$$\phi'(x) = \frac{1}{x} - \frac{1}{n} ,$$

the integrand is maximized at $x_{\max} = n$. At that point we have

$$\phi''(x_{\max}) = -\frac{1}{n^2} ,$$

and (A.32) gives

$$n! = \left(1+O(1/n)\right)\sqrt{\frac{2\pi}{n\left|\phi''(x_{\max})\right|}}\,\mathrm{e}^{n\phi(x_{\max})} = \left(1+O(1/n)\right)\sqrt{2\pi n}\,n^n\,\mathrm{e}^{-n} .$$